SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Dominik Knez, Marko Kukor, Ante Dumančić

Vizualizacija najkraćih ruta vremena putovanja i energetske potrošnje

Zagreb, 2024.

Ovaj rad izrađen je na Fakultetu prometnih znanosti Sveučilišta u Zagrebu na Zavodu za inteligentne transportne sustave na katedri za Primjenjeno računalstvo pod vodstvom prof. dr.sc. Tončija Carića i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2023/2024.

Sadržaj

1 UVOD

U svjetlu rastućih globalnih energetskih zahtjeva i sve većih ekoloških izazova, iznimno je važno razviti održiva rješenja u svim sektorima, posebice u prometu. Većina energetskih potreba u prometu trenutno se zadovoljava korištenjem fosilnih goriva, čije sagorijevanje značajno doprinosi emisiji stakleničkih plinova, što ima ozbiljne ekonomske i ekološke posljedice.

Ključno rješenje za ublažavanje ovih problema leži u prijelazu na čiste izvore energije. Električna energija, vođena tehnološkim inovacijama u području baterijskih tehnologija, ističe se kao predvodnik u cestovnom prometu. Unatoč kontinuiranom unaprjeđenju performansi električnih vozila, ograničeni domet baterija ostaje izazovna prepreka.

Inteligentni transportni sustavi (ITS) igraju ključnu ulogu u unapređenju prometnih sustava kroz integraciju naprednih upravljačkih i informacijsko-komunikacijskih tehnologija. U sklopu ovog istraživanja razvijena je aplikacija za određivanje prostornog dosega vozila u gradu Zagrebu, koristeći modeliranje cestovne mreže kao graf i implementaciju algoritama poput Dijkstrinog i Bellman-Fordovog te Fibonaccijeve hrpe za efikasnu analizu velikih skupova podataka.

Interaktivno grafičko sučelje omogućuje korisnicima vizualizaciju kontura prostornog dosega uzimajući u obzir vrijeme polaska te specifična vremenska i energetska ograničenja, čime se olakšava informirano planiranje putovanja. Prometne rute variraju u različita doba dana, zbog čega je ključno uzeti u obzir prometne gužve i uvjete na cestama koji se mogu značajno mijenjati tijekom dana.

Implementacija ove tehnologije pruža intuitivni alat za efikasno planiranje održivih putovanja, čime se značajno doprinosi mobilnosti i energetskoj učinkovitosti u urbanim područjima poput Zagreba. U kontekstu sve veće prisutnosti električnih vozila u Europi i Hrvatskoj, te potrebe za smanjenjem emisija stakleničkih plinova, ovaj sustav ima široku primjenu i potencijal za daljnji razvoj urbanog prometa.

Stoga, ovaj rad ima za cilj razviti inovativni sustav za analizu prometnih tokova koji uzima u obzir kriterije vremenske efikasnosti i energetske učinkovitosti, pružajući vrijedan alat za planiranje putovanja u urbanim sredinama, s posebnim naglaskom na vizualizaciju dometa vozila u izrađenom grafičkom sučelju.

1.1 Ciljevi, metode i koraci izrade rada

Cilj ovog istraživanja je postizanje značajnih rezultata kroz precizno definirane metode i korake, koji uključuju sljedeće faze:

1. Modeliranje prometne mreže i priprema podataka: Prvi korak istraživanja obuhvaća modeliranje grafa prometne mreže koristeći dostupne podatke iz projekta SORDITO. Ovi podaci uključuju temeljne informacije o cestovnim segmentima i prometnim karakteristikama, ključne za razvoj algoritama za pronalazak optimalnih ruta.
2. Razvoj algoritama za optimalno usmjeravanje: Druga faza istraživanja fokusirana je na razvoj modificiranog Dijkstrinog algoritma za pronalazak najkraćih vremenskih ruta. Osim toga, implementacija Bellman-Ford algoritma pripremit će graf za primjenu Dijkstrinog algoritma kako bi se pronašle energetski optimalne rute.
3. Primjena Fibonacci heap strukture za optimizaciju performansi: Treća faza istraživanja uključuje implementaciju Fibonacci heap strukture radi smanjenja vremena izvođenja algoritama. Ova napredna struktura ubrzat će manipulaciju i ažuriranje prioriteta čvorova grafa, što je ključno za brze izračune najkraćih ruta na velikim cestovnim mrežama.
4. Generiranje kontura prostornog dosega: Ova faza obuhvaća stvaranje kontura prostornog dosega korištenjem različitih metoda. Konture, koje prikazuju prostorni doseg na temelju vremena ili energetske potrošnje, pripremaju se za vizualizaciju u grafičkom sučelju.
5. Izrada interaktivnog web grafičkog sučelja: Peta faza istraživanja fokusira se na razvoj intuitivnog web sučelja koje omogućuje korisnicima odabir početne lokacije, vremena polaska, te vizualizaciju kontura prostornog dosega u odnosu na vrijeme putovanja ili energetsku potrošnju. Ovo sučelje omogućuje korisnicima da vizualiziraju kako različita vremenska ili energetska ograničenja utječu na doseg vozila. Rezultati se prikazuju kao prostorne karte koje jasno prikazuju doseg za odabrane scenarije, što korisnicima olakšava planiranje putovanja.
6. Analiza rezultata: Posljednja faza istraživanja uključuje analizu vremenskih trajanja putovanja i energetske potrošnje za različite scenarije. Ova analiza pružit će uvid u praktičnu primjenu razvijenog sustava u urbanim okruženjima kao što je grad Zagreb.

1.2 Struktura rada

Naslov rada je "Vizualizacija najkraćih ruta vremena putovanja i energetske potrošnje". Rad je organiziran u sedam tematskih cjelina:

1. Uvod
2. Opis podataka cestovne mreže
3. Pronalazak optimalnog puta na grafu
4. Fibonaccijeva hrpa
5. Određivanje prostornog dosega
6. Grafičko sučelje
7. Zaključak
8. Zahvale.

U drugoj cjelini rada detaljno su opisani podatci o cestovnoj mreži korišteni za izradu istraživanja, uključujući proces učitavanja tih podataka u računalni program.

Treća cjelina fokusira se na modeliranje cestovne mreže kao grafa te na pronalazak optimalnog puta na grafu. Prikazan je početni problem sedam mostova Königsberga, nakon čega slijedi detaljno objašnjenje teorije grafova. Poseban naglasak stavljen je na primjenu Bellman-Ford i Dijkstrinog algoritma za rješavanje problema optimalnog puta.

Četvrto poglavlje analizira primjenu Fibonaccijeve hrpe kao optimizacijske tehnike za ubrzanje rada Dijkstrinog algoritma. Objašnjeni su principi i operacije koje se primjenjuju na Fibonaccijevoj hrpi, zajedno s rezultatima implementacije.

Peto poglavlje opisuje određivanje prostornog dosega, uključujući identifikaciju najbližeg vrha, filtriranje ruta prema ograničenjima kao što su vrijeme i energija, te generiranje konkavnih ili konveksnih kontura na karti koristeći dobivene koordinate.

Šesto poglavlje posvećeno je dizajnu grafičkog korisničkog sučelja web aplikacije, koje omogućuje vizualizaciju kontura prostornog dosega najkraćih ruta vremena putovanja i energetske potrošnje. Analizirana je struktura web aplikacije, s naglaskom na frontend dio. Detaljno je objašnjeno kako je organiziran i kako funkcionira korisnički sučelje, uz opisan rad web aplikacije.

Sedmo poglavlje rada zaključuje istraživanje, sažima ključne rezultate i ističe njihovu relevantnost.

Na kraju rada navedene su zahvale svima koji su doprinijeli istraživanju, popis literature, kao i popisi slika, tablica. Također su priloženi sažetak rada na hrvatskom i engleskom jeziku te kratki životopis autora.

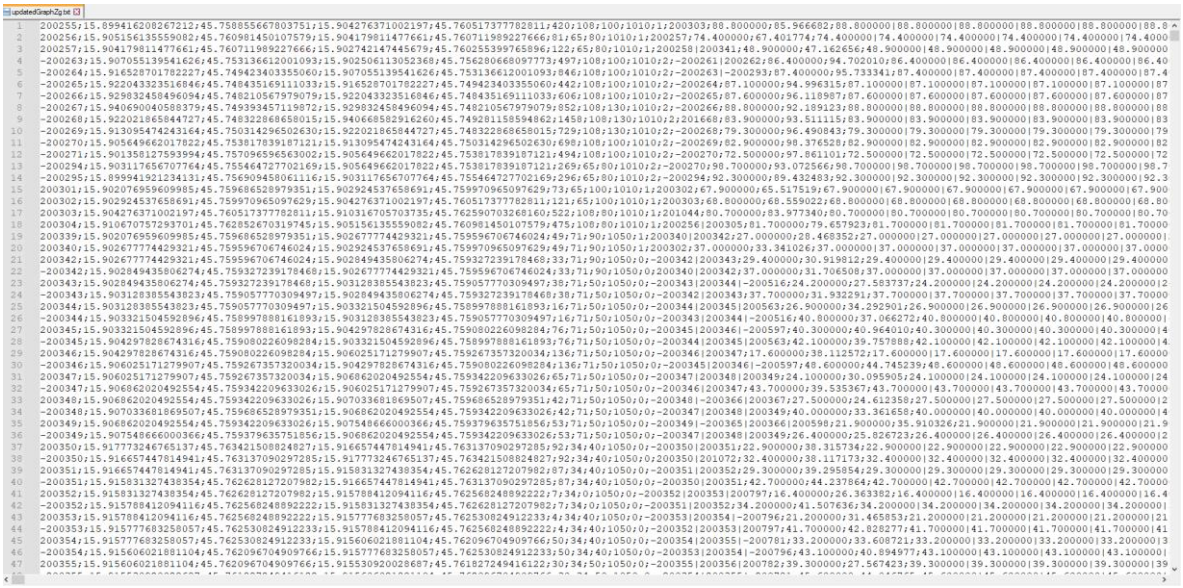
2 OPIS PODATAKA CESTOVNE MREŽE

Za rješavanje problema pronalaska vremenski najkraćeg i energetski optimalnog puta, potrebni su specifični podaci. U okviru ovog rada, korišteni su podaci prikupljeni kroz projekt SORDITO. Prikupljanje podataka trajalo je pet godina, od kolovoza 2009. do listopada 2014. godine. Tijekom tog razdoblja, koristilo se 4908 vozila opremljenih navigacijskim uređajima. Ovi uređaji slali su signale otprilike svakih 100 metara dok su vozila bila u pokretu i svakih 5 minuta kada su bila zaustavljena. Kao rezultat tog prikupljanja, dobiveno je 6.55 milijardi zapisa o kretanju vozila, s ukupnom pohranom podataka od 320 GB, (1). U ovom radu analizirana je cestovna mreža šire okolice grada Zagreba, prikazana na **Slika 1**, dok dodijeljena .txt datoteka ima veličinu od 402 MB, (2).

Slika 1. Razmatrana cestovna mreža

2.1 Struktura podataka

Prikupljeni GPS podaci pohranjeni su u CSV formatu (engl. Comma Separated Values). Na **Slika 2** prikazan je uzorak tih podataka. Svaki redak u CSV datoteci sadrži informacije o jednom cestovnom segmentu, poznatom kao link. Dodijeljena datoteka obuhvaća ukupno 43.992 redaka, što odgovara broju cestovnih segmenata.



Slika 2. Isječka podataka iz .txt datoteke u CSV formatu

Podaci, odnosno atributi za svaki pojedini link, smješteni su u jednom retku te su međusobno odvojeni točka-zarezom (;). Redoslijed tih podataka prikazan je u **Tablica 1**

Tablica 1. Popis atributa linka

Atribut	Opis
ID	ID cestovnog segmenta (cijeli broj)
X_A	Geografska duljina početne točke linka (A)
Y_A	Geografska širina početne točke linka (A)
X_B	Geografska duljina početne točke linka (B)
Y_B	Geografska širina početne točke linka (B)
L	Duljina linka u metrima
v_m	Brzina linka u km/h (statična brzina) postavljena od tvrtke Mireo d.d.
v_{og}	Ograničenje brzine linka u km/h
TL	Tip linka: 1010 su autoceste, a 1080 kvartovske male ceste, gradacija ide po 10 između tih vrijednosti
ZS	Zastavica smjera: moguće vrijednosti 0,1,2 ili 3. 0 – po linku se može putovati u dva smjera – u tom slučaju u datoteci postoje dva retka za link, jedan za pozitivni i jedan za negativni ID linka. 1 – po linku se putuje u jednom smjeru, a ID linka je pozitivna vrijednost, i to od točke A prema točci B 2- po linku se putuje u jednom smjeru, a ID link je negativne vrijednosti, i to od točke B prema točci A

	3- sve isto kao i za vrijednost 0, ali 3 predstavlja da je cesta zbog nekog razloga zatvorena
SL	Susjedni linkovi: između dvije susjedne točke-zarez (;) nalazi se lista susjednih linkova, pri čemu su ID-evi linkova međusobno odvojeni vertikalnom crtom (|)
v_{free}	Brzina slobodnog toka u km/h
v_{avg}	Prosječna brzine linka u km/h
P_v	Profil brzine u km/h: između dvije susjedne točke-zarez (;) nalazi se 288 vrijednosti brzina međusobno odvojenih vertikalnom crtom (|). Svaka od vrijednosti predstavlja prosječnu brzinu unutar pojedinog peto-minutnog intervala. Prva vrijednost predstavlja brzinu unutar intervala 00:00-00:05, druga unutar intervala 00:05-00:10, a zadnja unutar intervala 23:55-00:00.
E_{avg}	Prosječna potrošnja energije na linku u kWh
P_e	Profil energije u kWh: između dvije susjedne točke-zarez (;) nalazi se 288 vrijednosti potrošnje energije međusobno odvojene vertikalnom crtom (|). Svaka od vrijednosti predstavlja prosječnu potrošnju energije unutar pojedinog peto-minutnog intervala. Prva vrijednost predstavlja potrošnju energije unutar intervala 00:00-00:05, druga unutar intervala 00:05-00:10, a zadnja unutar intervala 23:55-00:00.

2.2 Izračun potrošnje energije

Budući da podaci o potrošnji energije nisu bili dostupni tijekom prikupljanja, potrebno je izvršiti procjenu te vrijednosti. U tu svrhu koristi se jednadžba prikazana na **Slika 3**, koja omogućuje izračun sile F potrebne za ubrzavanje vozila i prevladavanje otpora nagiba, kotrljanja i zraka. Parametri korišteni u ovoj jednadžbi navedeni su u **Tablica 2**. Ako je rezultirajuća sila $F \geq 0$, to znači da vozilo ubrzava i potrebna mu je snaga za kretanje. S druge strane, ako je $F < 0$, vozilo usporava (koči) ili se kreće nizbrdo, pri čemu se energija vraća u bateriju, (1).

$$F = \underbrace{mg\sin\alpha}_{\text{Nagib}} + \underbrace{c_r mg\cos\alpha}_{\text{Kotrljanje}} + \underbrace{0.5 c_d \rho A v^2}_{\text{Zrak}} + \underbrace{ma}_{\text{Ubrz.}}$$

Slika 3. Jednadžba za izračun sile, *(3)*

Tablica 2. Parametri jednadžbe za izračun sile, *(4)*

Oznaka	**Jedinica**	**Opis**
m	kg	Masa vozila (praznog)
a	m/s^2	Ubrzanje vozila
v	m/s	Brzina vozila
g	m/s^2	Gravitacijska konstanta
f		Faktor mase rotirajućih dijelova vozila
α	rad	Nagib prometnice
c_r		Koeficijent trenja kotrljanja
c_d		Koeficijent otpora zraka
ρ	kg/m^3	Gustoća zraka
A	m^2	Prednja površina vozila

Kako bi se izračunala potrošnja energije, potrebno je uzeti u obzir specifične parametre za odabrani električni automobil. U ovom primjeru koristi se Nissan Leaf 2014, a njegove tehničke specifikacije prikazane su u **Tablica 3**.

Tablica 3. Karakteristike vozila Nissan Leaf 2014, *(1)*

Opis	**Oznaka**	**Vrijednost**
Kapacitet baterije	Q	24 kWh
Masa	m	1145 kg
Koeficijent trenja kotrljanja	c_r	0.008
Koeficijent otpora zraka	c_d	0.35
Prednja površina vozila	A	1.9 m^2
Faktor mase rotirajućih dijelova vozila	f	1.01
Koeficijent motora i pogonskog sklopa	μ_m	0.9
Koeficijent između pogona i motora	μ_g	0.8
Pomoćna snaga	P_0	450 W
Minimalna brzina za vraćanje energije	v_{min}	10 km/h

Gustoća zraka ρ postavljena je na 1.2 kg/m³, dok je gravitacijsko ubrzanje g definirano kao 9.81 m/s². Vrijednosti brzine v, ubrzanja a, i nagiba α nisu vezane uz tehničke specifikacije vozila, već ovise o karakteristikama samih cestovnih segmenata. Brzine v korištene u izračunu temelje se na podacima prikupljenim tijekom projekta SORDITO. Zbog malih razlika u vrijednostima brzina između susjednih intervala vrijednosti ubrzanja preuzete su iz studije o vozilu Nissan Leaf, pri čemu su dodijeljene odgovarajućim intervalima brzine, kako je

prikazano u **Tablica 4**. Za izračun nagiba prometnih segmenata koriste se digitalni elevacijski podaci Europske službe Copernicus. Svaki cestovni segment ima određenu razliku u nadmorskoj visini (Δh) i zračnu udaljenost (d). Nagib se izračunava pomoću jednadžbe: $\alpha = arctan\frac{\Delta h}{d}$, (1,2).

Tablica 4. Akceleracija prema intervalima brzine za Nissan Leaf 2014, *(1)*

Interval brzine $[km/h]$	Akceleracija $[m/s^2]$
$[0, 30\rangle$	0.61
$[30, 51\rangle$	0.53
$[51, 72\rangle$	0.37
$[72, 93\rangle$	0.41
$[93, 102\rangle$	0.28
$[102, \infty\rangle$	0.05

Nakon dobivanja rezultirajuće sile, koristi se jednadžba prikazana na **Slika 4** za izračun snage baterije. U toj jednadžbi, μ_m označava koeficijent prijenosa između elektromotora i pogonskog sustava, dok μ_g predstavlja omjer pretvorbe mehaničke energije na kotačima u kemijsku energiju pohranjenu u bateriji. Bitno je napomenuti da se energija vraća u bateriju samo ako je sila F manja od nule i brzina veća od v_{min}. Kada se izračuna snaga baterije, može se izračunati trenutna potrošnja energije E pomoću jednadžbe $E = P_b\Delta t$, gdje Δt označava vrijeme putovanja na cestovnom segmentu, (1,2).

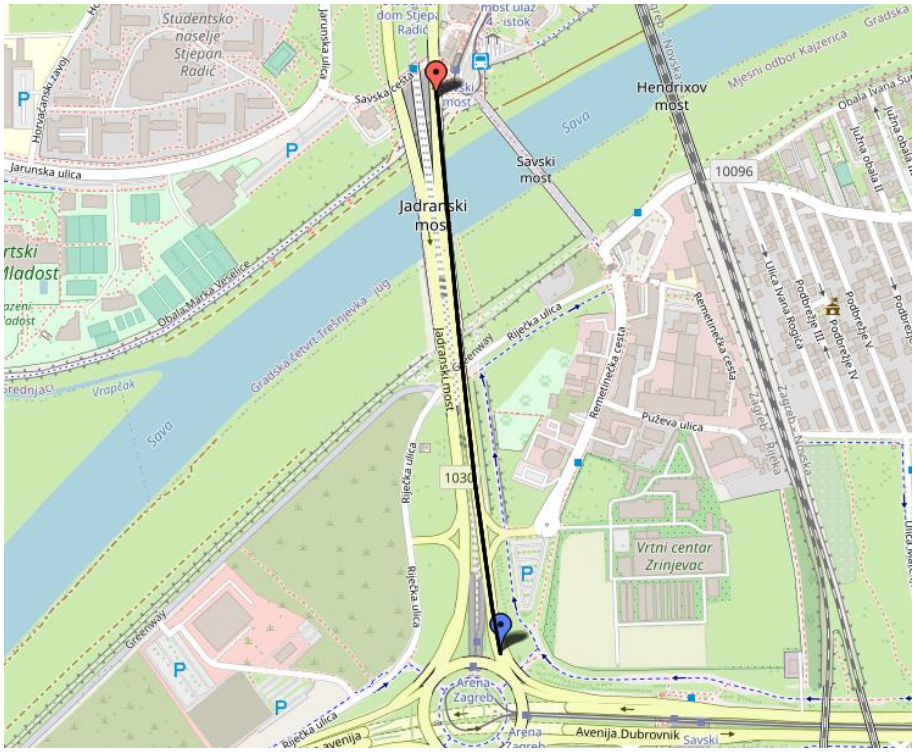
$$P_b = \begin{cases} \frac{Fv}{\mu_m} + P_0, & \text{if } F \geq 0 \\ \begin{cases} 0, & \text{if } v \leq v_{min} \\ \mu_g F v + P_0, & \text{else} \end{cases}, & \text{if } F < 0 \end{cases}$$

Slika 4. Jednadžba za izračun snage baterije, *(3)*

2.3 Primjer cestovnog segmenta

Za bolje razumijevanje, prikazan je link na Jadranskom mostu na **Slika 5** kao primjer određenog cestovnog segmenta u datoteci. Identifikacijski broj tog segmenta je -214695, a njegova duljina iznosi 340 metara. Ograničenje brzine na ovom segmentu postavljeno je na 60

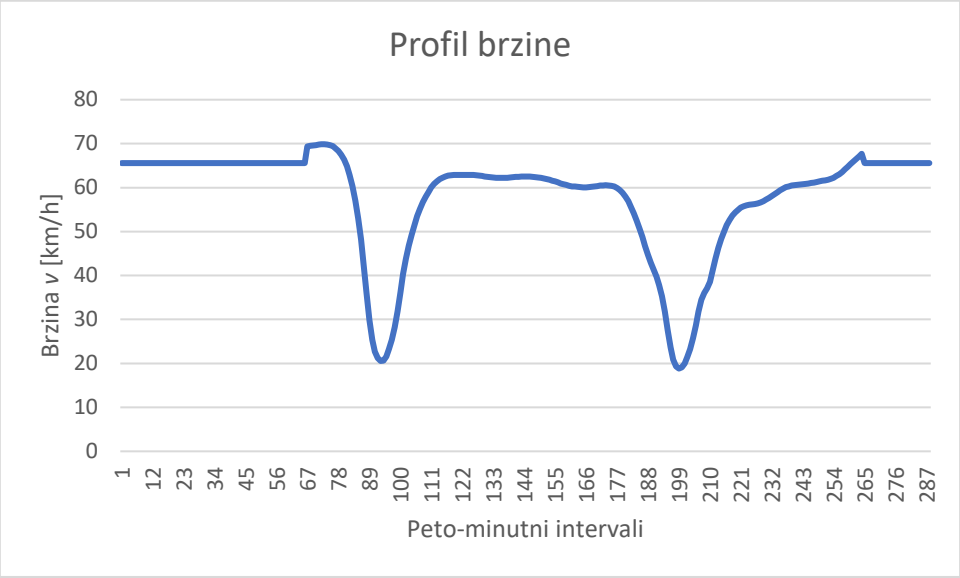
km/h. Tip cestovnog segmenta je označen kao 1040, što ukazuje da se radi o glavnoj cesti, dok zastavica smjera s brojem 2 ukazuje da se putovanje tim segmentom odvija samo u jednom smjeru. Segment ima samo jedan susjedni link identificiran brojem -214694. Prosječna brzina kretanja vozila na Jadranskom mostu iznosi 53.89 km/h, dok je prosječna potrošnja energije na tom segmentu 0.072 kWh, (2).



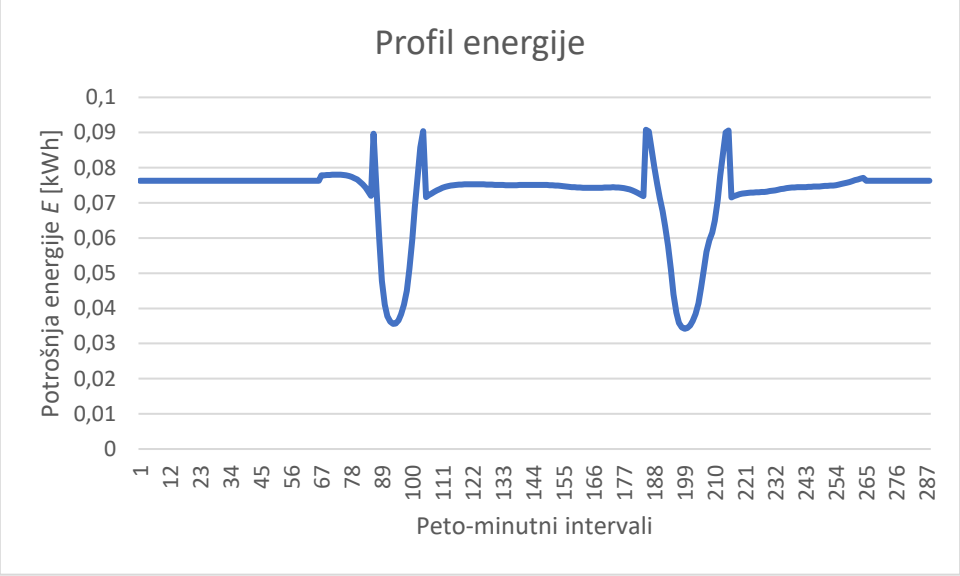
Slika 5. Primjer cestovnog segmenta -214695

Za potrebe određivanja vremenski-ovisnog vremenskog najkraćeg ili energetski optimalnog puta bitno je istaknuti izračunati profil brzine i profil potrošnje energije za svaki cestovni segment, koji se sastoje od 288 vrijednosti, gdje svaka predstavlja prosječnu brzinu ili prosječnu potrošnju energije u petominutnom intervalu u danu. Na **Slika 6** prikazan je profil brzine na Jadranskom mostu za dan ponedjeljak, gdje su primjetni padovi brzine oko 90. i 200. vrijednosti. Ti padovi odgovaraju vršnim satima prometa, otprilike oko 7:30 i 16:30 sati, kada često dolazi do prometnih zagušenja, što usporava kretanje vozila u usporedbi s brzinom u slobodnom toku. Slično tome, na **Slika 7** prikazan je profil energetske potrošnje, gdje se također primjećuje smanjenje potrošnje energije za ista vremenska razdoblja. To je posljedica sporijeg

kretanja vozila uslijed prometnih zagušenja tijekom vršnih sati, što rezultira nižom potrošnjom energije, (2).



Slika 6. Profil brzine na Jadranskom mostu



Slika 7. Profil energije na Jadranskom mostu

2.4 Učitavanje podataka

Za učitavanje podataka u programskom jeziku C# korišten je Visual Studio 2022. Prvi korak je bio stvaranje klase koja sadrži sve atribute navedene u **Tablica 1**. Nakon definiranja klase, za čitanje podataka iz .txt datoteke koristila se klasa StreamReader, pristupajući joj putem imenskog prostora "using System.IO". Zatim su učitani svi redci datoteke te je za svaki redak stvoren objekt klase s odgovarajućim vrijednostima atributa.

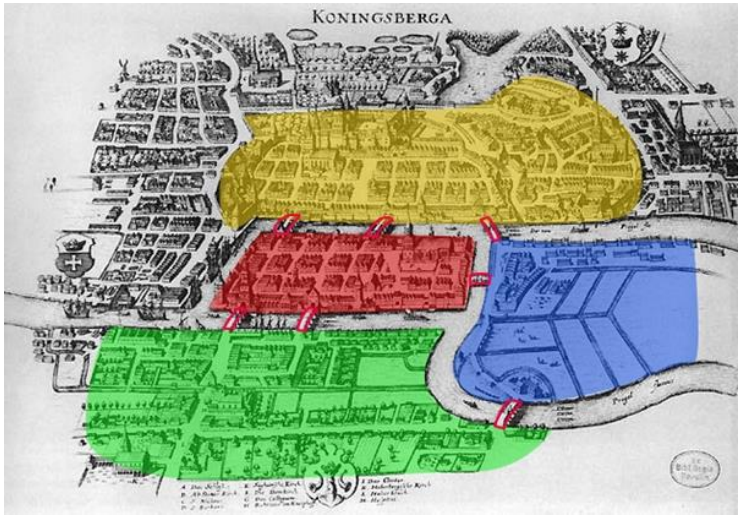
Svaki stvoreni objekt klase je potom pohranjen u rječnik (engl. Dictionary). Rječnik koristi jedinstveni ključ za svaki element radi brzog pristupa, (5).

3 PRONALAZAK OPTIMALNOG PUTA NA GRAFU

Prometna mreža se često modelira kao skup cesta i raskrižja, gdje su ceste reprezentirane bridovima, a raskrižja čvorovima, tvoreći matematički graf. Pronalaženje najboljeg puta na grafu zahtijeva primjenu teorije grafova.

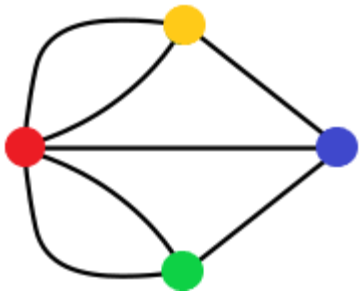
3.1 Problem sedam mostova Königsberga

U 18. stoljeću, u gradu Königsbergu u Pruskoj, postojalo je sedam mostova preko rijeke koja je prolazila kroz grad, što je prikazano na **Slika 8**. Građani Königsberga su stvorili igru u kojoj je cilj bio pronaći put kroz grad koji bi uključivao prelazak preko svakog mosta točno jednom, te se vratiti na početnu točku. Ovaj problem je izazvao interes švicarskog matematičara Leonarda Eulera, koji je 1736. godine uspješno riješio izazov sedam mostova Königsberga, (6).



Slika 8. Raspored mostova Königsberga, *(6)*

Euler je formulirao pretpostavku da je unutarnji redoslijed kretanja kroz kopnene površine nebitan, fokusirajući se isključivo na redoslijed prelaska mostova. Ova ideja omogućila mu je da modelira topologiju grada kao graf, prikazan na **Slika 9**, gdje su kopnene površine predstavljene čvorovima, a mostovi bridovima, (7,8).



Slika 9. Prikaz problema sedam mostova Königsberga pomoću grafa, *(8)*

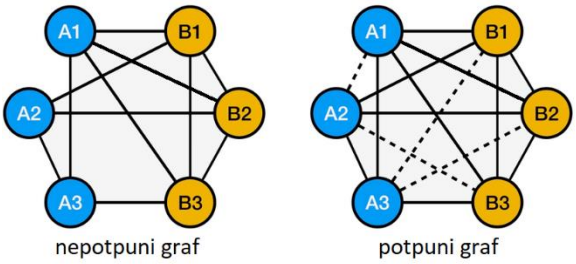
Dalje je Euler zaključio da je moguće pronaći traženu rutu ovisno o stupnjevima čvorova, gdje stupanj čvora označava broj bridova koji ga spajaju. To je dovelo do definicije Eulerovog puta, koji postoji samo ako graf ima nula ili dva čvora s neparnim stupnjem. S obzirom da graf koji odgovara problemu sedam mostova Königsberga ima četiri čvora s neparnim stupnjem, Eulerov put u ovom slučaju nije moguć. Ovaj problem smatra se jednim od pionirskih problema teorije grafova, (7,8).

3.2 Teorija grafova

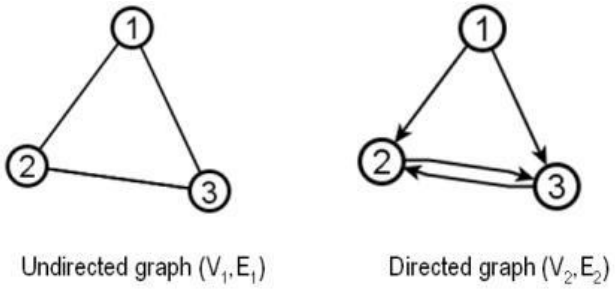
Razmatranje grafova kao matematičkog područja naziva se teorija grafova. Graf se sastoji od skupa vrhova i skupa bridova. Grafovi se mogu klasificirati prema različitim svojstvima kao što su potpuni i nepotpuni, usmjereni i neusmjereni težinski simetrični i asimetrični grafovi, što je prikazano na **Slika 10**, **Slika 11** i **Slika 12**.

U potpunom grafu, svaki čvor je povezan s ostalim čvorovima bridom, dok u nepotpunom grafu neki čvorovi nemaju bridove među sobom. U usmjerenom grafu, bridovi se promatraju kao usmjerene veze između vrhova A i B, što može biti u obliku A->B ili B->A. Nasuprot tome, u neusmjerenom grafu, brid se promatra kao veza koja nema određeni smjer, (2).

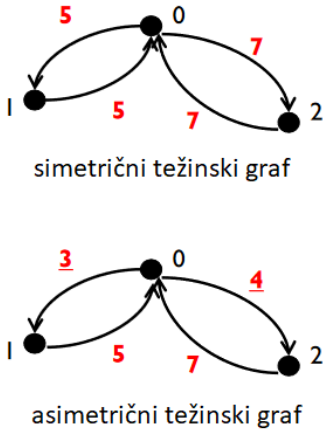
U usmjerenom grafu, težinski graf može biti simetričan ili asimetričan. U simetričnom težinskom grafu, težina bridova između dva vrha je ista bez obzira na smjer u kojem se promatraju. S druge strane, u asimetričnom težinskom grafu, težine bridova mogu biti različite ovisno o smjeru promatranja. Kod neusmjerenog grafa, težinski graf uvijek je simetričan, (9,10).



Slika 10. Prikaz potpunog i nepotpunog grafa, *(11)*



Slika 11. Prikaz usmjerenog i neusmjerenog grafa, *(12)*



Slika 12. Prikaz simetričnog i asimetričnog grafa, *(10)*

Za potrebe ovog istraživanja, graf je modeliran prema stvarnoj cestovnoj mreži grada Zagreba i predstavlja nepotpuni usmjereni težinski asimetrični graf. Ovaj graf sadrži 43992 vrhova i 111350 bridova.

U ovom grafu, vrhovi predstavljaju cestovne segmente, dok bridovi označavaju mogućnost putovanja između segmenata u specifičnom smjeru. Graf je usmjeren, što znači da svaki brid ima određeni smjer, a atribut svakog cestovnog segmenta (linka) pokazuje je li putovanje moguće u oba smjera, samo u jednom smjeru, ili je putovanje potpuno zatvoreno.

Graf je također nepotpun, što znači da ne postoji veza između svih mogućih parova vrhova. Svaki vrh sadrži popis susjednih vrhova (linkova) do kojih se može putovati, čime se jasno prikazuje da nisu svi vrhovi međusobno povezani.

Kao težinski graf, svaki brid u ovom modelu ima pridruženu težinu koja može predstavljati različite faktore poput udaljenosti, vremena putovanja ili energetske potrošnje. Asimetričnost ovog grafa znači da težina putovanja između dva segmenta može varirati ovisno o smjeru putovanja. Na primjer, energetska potrošnja za putovanje od vrha A do vrha B može biti različita od potrošnje u suprotnom smjeru, pri čemu neki segmenti mogu imati pozitivnu energetsku potrošnju u jednom smjeru i negativnu u drugom.

3.3 Algoritam za pronalazak optimalnog puta

U teoriji grafova, jedan od ključnih problema je pronalazak najkraćeg puta između zadanih vrhova grafa. Dijkstrin algoritam je poznat po svojoj učinkovitosti u pronalaženju najkraćeg puta od jednog vrha do svih ostalih vrhova u grafu. Ovaj algoritam koristi težine bridova kako bi odredio najmanji ukupni trošak puta. Kod traženja najkraćeg vremenskog puta, Dijkstrin algoritam se lako primjenjuje jer težine bridova predstavljaju vremenske ili prostorne udaljenosti, koje su uvijek pozitivne. Međutim, kada se radi o pronalaženju energetski optimalnog puta za električna vozila, težine bridova se izražavaju kao potrošnja energije. U ovom kontekstu, potrošnja energije može biti pozitivna (trošak energije za kretanje) ili negativna (energija koja se regenerira tijekom vožnje nizbrdo ili pri kočenju, vraćajući se u bateriju). Ovakva svojstva predstavljaju izazov za Dijkstrin algoritam, koji je koncipiran da radi samo s pozitivnim vrijednostima težina, (13,14).

3.3.1 Dijkstra algoritam s dinamičkim vremenskim težinama

Dijkstrin algoritam, koji je prvi put opisao nizozemski računalni znanstvenik Edsger W. Dijkstra 1959. godine, koristi se za pronalaženje najkraćih putova u grafovima s pozitivnim težinama bridova. Klasična verzija ovog algoritma, prikazana u **Algoritmu 1**, započinje inicijalizacijom svih vrhova u grafu tako da se svi vrhovi postave na beskonačnost, osim početnog vrha kojem se dodjeljuje težina nula. Tijekom svake iteracije, algoritam ažurira težine susjednih vrhova i bilježi prethodne vrhove kako bi omogućio rekonstrukciju najkraćeg puta, (15,16).

U proširenoj verziji Dijkstrinog algoritma, koja uključuje vremenski ovisnu komponentu, težine između vrhova predstavljaju vremena putovanja koja se dinamički prilagođavaju ovisno o dobu dana. U ovoj verziji, vrijeme putovanja između vrhova izračunava se kao kvocijent duljine linka i brzine kretanja specifične za određeno vremensko razdoblje.

Početno vrijeme putovanja, izraženo u sekundama, igra ključnu ulogu u ovoj verziji algoritma. Ovo vrijeme određuje koji će se indeks koristiti iz profila brzine, pri čemu svaki indeks odgovara petominutnom vremenskom intervalu. Budući da dan ima 1440 minuta, to rezultira s 288 petominutnih intervala. Početno vrijeme putovanja pretvara se u sekunde kako bi se odredio odgovarajući indeks profila brzine.

Algoritam započinje tako što dodaje sve vrhove grafa u listu neobrađenih vrhova. Početni vrh postavlja se s težinom nula, dok se ostalim vrhovima dodjeljuje težina beskonačnost. Svakom vrhu dodjeljuje se atribut prethodnog vrha na prazninu (engl. Null). Tijekom iteracija, algoritam pregledava sve neobrađene susjedne vrhove trenutnog vrha. Za svaki susjedni vrh izračunava se ukupno vrijeme putovanja, koje uključuje vrijeme trenutnog vrha i vrijeme putovanja između trenutnog i susjednog vrha. Vrijeme putovanja između vrhova izračunava se prema profilu brzine koji se određuje pomoću početnog vremena putovanja.

Ako novo izračunato vrijeme putovanja do susjednog vrha bude manje od trenutnog vremena za taj vrh, ažurira se vrijeme i postavlja trenutni vrh kao prethodni vrh. Početno vrijeme putovanja se zatim povećava za izračunato vrijeme putovanja do susjednog vrha. Novi indeks profila brzine određuje se na temelju ažuriranog početnog vremena, što omogućava korištenje preciznijih podataka o brzini za svaki sljedeći korak algoritma, (1).

Nakon što su svi susjedni vrhovi obrađeni, trenutni vrh uklanja se s liste neobrađenih vrhova, a zatim se traži novi neobrađeni vrh s najmanjom težinom, što je prikazano u **Algoritmu 2**. Postupak se ponavlja sve dok lista neobrađenih vrhova ne postane prazna. Na kraju, Dijkstrin algoritam pruža najkraće težinske udaljenosti od početnog vrha do svih drugih vrhova u grafu.

Da bi se prikazao put od početnog vrha do bilo kojeg drugog vrha, koristi se petlja koja počinje od odabranog vrha i iterativno se pomiče prema početnom vrhu, prateći prethodne vrhove sve dok atribut prethodni vrh ne dostigne vrijednost null, koja označava početni vrh. Na taj način može se rekonstruirati cijeli put od početnog vrha do odabranog vrha, dodajući vrhove koji su prošli kroz petlju u listu putanje, kako je prikazano u **Algoritmu 3**, (4).

Korištenjem profila brzine za različite dijelove dana, algoritam može precizno izračunati najkraće putove uzimajući u obzir varijacije u brzini kretanja, kao što su one tijekom vršnih sati. Ova metoda omogućava prilagodbu najkraće rute uvjetima prometa u različitim vremenskim intervalima, čime se povećava točnost izračunavanja vremena putovanja. Algoritam kontinuirano ažurira indeks profila brzine prema ažuriranom početnom vremenu putovanja, što omogućava prilagodbu težina i optimalan izbor putova.

Algoritam 1. Dijkstra algoritam, (1)

Input: Graph G, start vertex v_0

```
 1: Q ← Initialize list of unprocessed vertices
 2: for each vertex v in graph G do
 3:     if v = v_0 then
 4:         value(v) ← 0
 5:     else
 6:         value(v) ← ∞
 7:     end if
 8:     preceding(v) ← None
 9:     Add v to Q
10: end for
11: while Q is not empty do
12:     u ← Remove vertex form Q with the lowest value
13:     for each unprocessed neighboring vertex v of vertex u do
14:         val_new ← value(u) + w_u,v
15:         if val_new < value(v) then
16:             value(v) ← val_new
17:             preceding(v) ← u
18:         end if
19:     end for
20: end while
```

Algoritam 2. Uklanjanje vrha s najmanjom vrijednošću, (2,4)

Input: List of unprocessed vertices Q

```
 1: value_min ← ∞
 2: u ← None
 3: for each vertex v in list Q do
 4:     if value(v) < value_min then
 5:         value_min = value(v)
 6:         u ← v
 7:     end if
 8: end for
 9: Return u
```

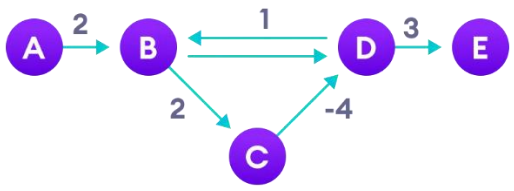
Algoritam 3. Dohvaćanje putanje, (2,4)

Input: Finish vertex v_{end}

```
1: Q ← Initialize list of vertices
2: v ← v_end
3: while preceding(v) ≠ null do
4:     Add v to Q
5:     v ← preceding(v)
6: end while
```

3.3.2 Bellman-Ford algoritam i Dijkstra algoritam s dinamičkim energetskim težinama

Pri primjeni Dijkstrinog algoritma na grafu koji sadrži negativne težine, postoji velika opasnost od dobivanja netočnih rješenja zbog postojanja negativnih ciklusa. Primjer jednog takvog ciklusa može se vidjeti na **Slika 13**, gdje Dijkstrin algoritam kreće iz čvora A(0) prema čvoru B(2). Algoritam ima opciju izbora između čvora C(4) i čvora D(3). Budući da čvor D ima manju težinu od čvora C, algoritam odabire čvor D(3) i nastavlja prema čvoru E, gdje se krajnja težina računa kao 6. Međutim, ovakav izbor može dovesti do pogrešnog rješenja jer da je algoritam odabrao čvor C(4), put do čvora D bi imao težinu -4, što bi rezultiralo težinom čvora D od 0. Konačno, težina čvora E bi bila 3, što je manje od težine dobivene Dijkstrinim algoritmom, (2,13).



Slika 13. Prikaz negativnog ciklusa, *(13)*

Kako bi se izbjegle potencijalne pogreške kod izvođenja Dijkstrinog algoritma na grafu s negativnim težinama, nužno je transformirati graf u graf s pozitivnim težinama. To se može postići korištenjem Johnsonove tehnike u kombinaciji s Bellman-Ford algoritmom. Postupak se detaljno opisuje u **Algoritmu 4**.

Bellman-Ford algoritam, iako sličan Dijkstrinom, razlikuje se u načinu obrade čvorova. Dok Dijkstrin algoritam fokusira se samo na neobrađene susjedne čvorove tijekom svake iteracije, Bellman-Ford algoritam prolazi kroz sve susjedne čvorove u svakoj iteraciji. Ova karakteristika omogućuje Bellman-Ford algoritmu da efikasno radi s grafom koji sadrži bridove s negativnim težinama. Zbog toga, Bellman-Ford algoritam ima složenost $O(E * V)$, gdje je E broj bridova, a V broj čvorova u grafu. To je veće od složenosti Dijkstrinog algoritma koja iznosi $O(E + V \log V)$. Iako je Bellman-Ford algoritam sporiji zbog ovog faktora, on se koristi samo jednom za pripremu grafa za Dijkstrin algoritam, (1).

Primjenom Johnsonove tehnike, dodaje se novi čvor Q u graf koji je povezan s svim ostalim čvorovima bridovima težine 0. Nakon toga se pokreće Bellman-Ford algoritam na modificiranom grafu, počevši od čvora Q. Rezultat algoritma su putevi minimalne težine od čvora Q do svakog drugog čvora u grafu, koji se zatim spremaju kao nove težine za bridove u grafu. Nakon završetka ovog koraka, čvor Q i svi njegovi bridovi se uklanjaju iz grafa. Na taj način se graf transformira u graf s pozitivnim težinama, čime je spreman za daljnje korištenje Dijkstrinim algoritmom, (1,17).

Algoritam 4. Bellman-Ford algoritam s Johnsonovom tehnikom, (1,2)

Input: Graph *G*, start vertex v_0

```
Q ← Initialize list of unprocessed vertices
Add v0 to G
for each vertex v in graph G do
    if v = v0 then
        value(v) ← 0
    else
        value(v) ← ∞
        neighbor(v0 )← v
    end if
    preceding(v) ← None
    Add v to Q
end for
while Q is not empty do
    u ← Remove vertex form Q with the lowest value
    for each neighboring vertex v of vertex u do
        val_new ← value(u) + w_u,v
        if val_new < value(v) then
            value(v) ← val_new
            preceding(v) ← u
        end if
     end for
end while
Remove v0 from G
```

Dijkstra algoritam s dinamičkim energetskim težinama sličan je Dijkstra algoritmu s dinamičkim vremenskim težinama, ali postoje dvije ključne razlike. Prva razlika odnosi se na vrstu težina koje se koriste. Umjesto vremena putovanja, u ovom algoritmu težine predstavljaju potrošnju energije, a vrijednosti se dobivaju iz profila energetske potrošnje. Druga razlika tiče

se metode izračuna novih težina za svaki susjedni vrh. Za svaki susjedni vrh, ukupna težina puta izračunava se kao zbroj težine trenutnog vrha, težine brida između trenutnog i susjednog vrha, te korekcije koju generira Bellman-Ford algoritam.

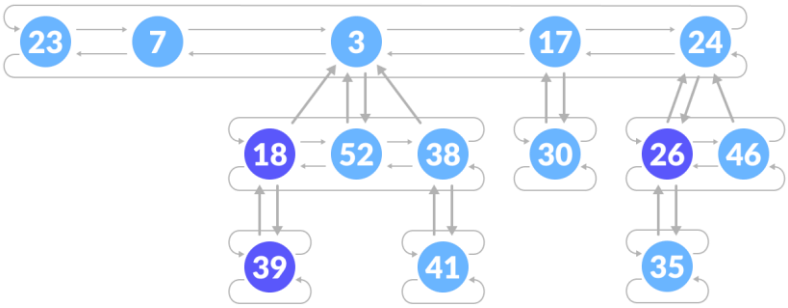
Dijkstrin algoritam se često implementira koristeći nesortirano polje za praćenje neobrađenih vrhova. Međutim, ova metoda ima svoj nedostatak jer ima složenost $O(n)$ za pronalaženje i izbacivanje vrha s najmanjom težinom, što može značajno usporiti algoritam. U ovom istraživanju, primijenjena je drugačija podatkovna struktura koja omogućuje efikasniji pristup neobrađenom vrhu s najmanjom težinom, (1,2,16). Korištenjem te strukture, postignuta je složenost algoritma $O(E + V \log V)$. Za graf slabe gustoće, gdje je broj bridova približno jednak broju vrhova $E \approx V$, što je slučaj u ovom istraživanju gdje svaki vrh u prosjeku ima 2.53 izlazna bridova, Dijkstrin algoritam ima složenost $O(V\ log\ V)$, (1).

4 FIBONACCIJEVA HRPA

Da bi se povećala učinkovitost i ostvarila $O(E + V \log V)$ složenost Dijkstrinog algoritma, ključno je riješiti problem uskog grla koje se pojavljuje prilikom pronalaženja i izbacivanja vrha s najmanjom težinom iz nesortiranog polja neobrađenih vrhova. Za postizanje optimalne složenosti i bržeg rada Dijkstrinog algoritma može se koristiti Fibonacci Heap struktura podataka (engl. Fibonacci Heap data structure), (2).

4.1 Struktura Fibonaccijeve hrpe

Fibonaccijeva hrpa je struktura podataka koja se sastoji od kolekcije stabala, gdje svako stablo zadovoljava svojstvo da je vrijednost svakog djeteta uvijek veća ili jednaka vrijednosti njegovog roditelja. Ova stabla su međusobno povezana kroz kružnu dvostruko povezanu listu, što je prikazano na **Slika 14**, (18,19).

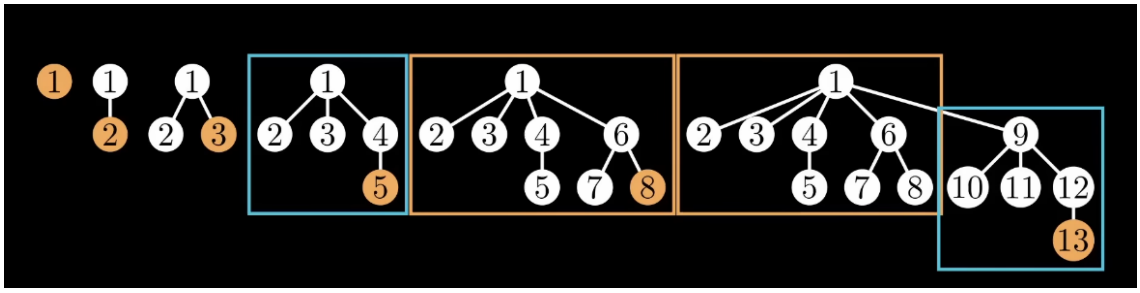


Slika 14. Fibonacci Heap struktura, *(19)*

Svako stablo u Fibonaccijevoj hrpi sastoji se od čvorova koji sadrže vrijednosti i reference na svoje dijete, roditelja i brata. Struktura Fibonaccijeve hrpe je fleksibilna jer stabla nemaju propisan oblik, što omogućuje izvođenje operacija na "lijen" način, čime se smanjuje složenost tih operacija. Međutim, u određenim situacijama hrpu je potrebno reorganizirati kako bi se održalo brzo vrijeme izvršavanja algoritma.

Nakon reorganizacije hrpe, kao što je prikazano na **Slika 15**, primjećuje se da broj elemenata u stablu stupnja n jednak je zbroju elemenata u stablu stupnja n−1 i n−2. Ovi brojevi, koji se

dobivaju zbrajanjem prethodna dva, nazivaju se Fibonaccijevi brojevi, odakle i potječe naziv Fibonaccijeva hrpa, (20).



Slika 15. Pojava Fibonaccijevih brojeva u uređenoj Fibonaccijevoj hrpi, *(20)*

4.2 Operacije na Fibonaccijevoj hrpi

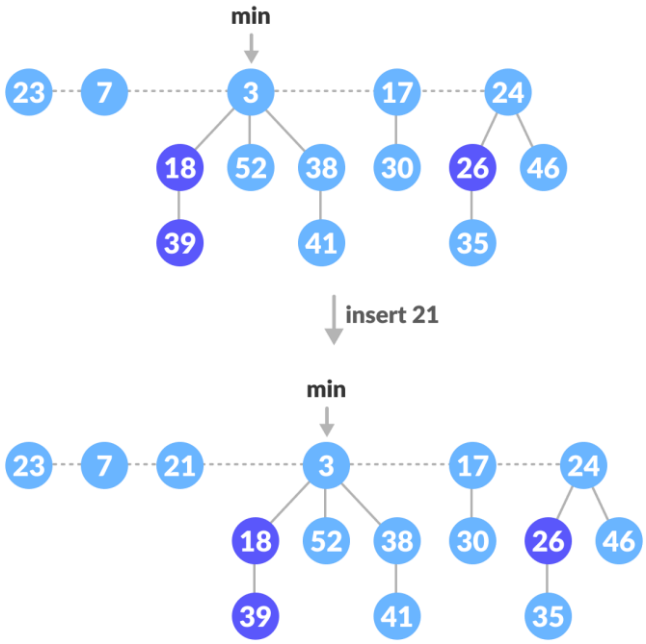
Kružna dvostruko povezana lista kojom su stabla povezana omogućuje brzu izvedbu određenih operacija.

Ključne operacije koje se mogu izvesti na Fibonaccijevoj hrpi su:

- Umetanje, engl. *Insert,*
- Pronađi minimum, engl. F*ind Min,*
- Spajanje, engl. *Union,*
- Izbaci minimum, engl. E*xtract Min,*
- Smanji vrijednost, engl. *Decrease Key,* (19).

4.2.1 Operacija Insert

Prvi korak pri operaciji *insert* je stvaranje novog čvora. Nakon toga se provjerava je li hrpa prazna. Ako jest, novi čvor se postavlja kao korijenski čvor i označava kao minimum. Ako hrpa nije prazna, novi čvor se dodaje na popis korijena, a minimalna vrijednost se ažurira. Na **Slika 16** prikazano je umetanje vrijednosti 21 u Fibonaccijevu hrpu. Operacija umetanja ima složenost $O(1)$, (19).



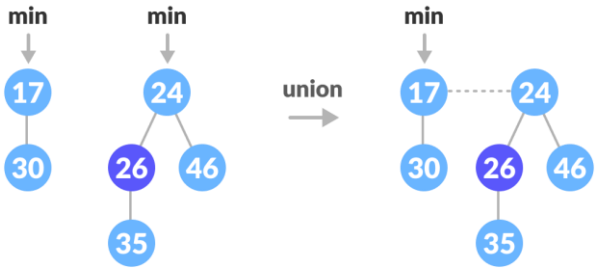
Slika 16. Operacija insert, *(19)*

4.2.2 Operacija Find Min

Fibonaccijeva hrpa omogućuje učinkovito dohvaćanje čvora s minimalnom vrijednosti jer hrpa uvijek ima pokazivač na taj čvor. Operacija *find min* se stoga svodi na jednostavan dohvat pokazivača na minimalni čvor, što se može ostvariti u samo jednoj liniji koda. Zbog toga je ova operacija vrlo brza i ima vremensku složenost $O(1)$, (19).

4.2.3 Operacija Union

Operacija *union* omogućuje spajanje dva ili više stabala u Fibonaccijevoj hrpi. Prvo se svi korijeni stabala spoje u jednu listu. Zatim se prolaskom kroz popis korijena ažurira pokazivač minimalne vrijednosti kako bi ukazivao na čvor s najmanjom vrijednošću. Ova operacija je prikazana na **Slika 17**, gdje se stabla s korijenima 17 i 24 spajaju, a pokazivač minimalne vrijednosti postavlja se na čvor s vrijednošću 17. Operacija *union* ima vremensku složenost $O(1)$, (19).



Slika 17. Operacija union, *(19)*

4.2.4 Operacija Extract Min

Jedan od ključnih koraka Dijkstrinog algoritma je pronalaženje i uklanjanje čvora s najmanjom težinom iz skupa. Operacija *extract min* obavlja upravo tu funkciju. Na **Slika 18**, postupak operacije *extract min* objašnjen je kroz detaljan primjer i korake od a do i. Prvi korak prikazuje početni izgled Fibonaccijeve hrpe.

Operacija *extract min* pronalazi i uklanja čvor s najmanjom vrijednosti iz hrpe, uz to provodi važnu operaciju reorganizacije hrpe. Koraci su sljedeći:

a) Prvi korak prikazuje početni izgled Fibonaccijeve hrpe.
b) Čvor s najmanjom vrijednosti se uklanja, a njegova djeca se dodaju u popis korijena. Pokazivač minimuma se zatim postavlja na sljedeći korijen u popisu. U primjeru se uklanja čvor 3, a njegova djeca (18, 52, 38) dodaju se u listu korijena. Pokazivač minimuma postavlja se na čvor 17.
c) Korijeni se potom sortiraju prema stupnju (broju djece), stvarajući stablo gdje svaki korijen ima jedinstveni stupanj. U primjeru, korijeni 23, 7, 21 i 52 grupiraju se u stupanj 0, korijeni 18, 38 i 17 u stupanj 1, a korijen 24 u stupanj 2.
d) Sljedeći korak uključuje spajanje korijena kako bi svaki imao jedinstveni stupanj. Korijeni 7 i 23 se spajaju jer imaju isti stupanj (0); čvor 7 ostaje u popisu korijena, a čvor 23 postaje dijete čvora 7. Stupanj čvora 7 se povećava na 1.
e) Korijeni 7 i 17 se spajaju jer imaju isti stupanj (1); čvor 17 postaje dijete čvora 7, čime stupanj čvora 7 postaje 2.

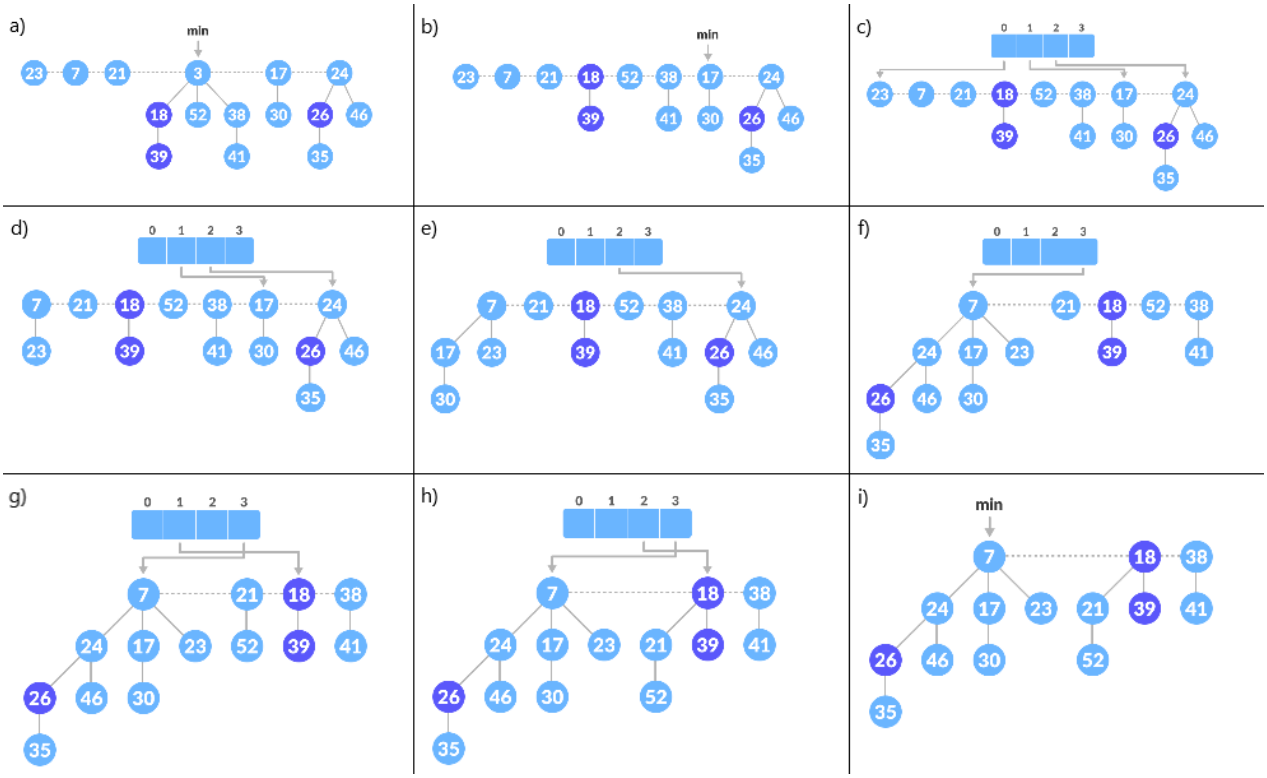
f) Korijeni 7 i 24 se spajaju jer imaju isti stupanj (2); čvor 24 postaje dijete čvora 7, povećavajući stupanj čvora 7 na 3.

g) Korijeni 21 i 52 se spajaju jer imaju isti stupanj (0); čvor 52 postaje dijete čvora 21, čime stupanj čvora 21 postaje 1.

h) Korijeni 18 i 21 se spajaju jer imaju isti stupanj (1); čvor 21 postaje dijete čvora 18, povećavajući stupanj čvora 18 na 2.

i) Nakon što svi korijeni imaju jedinstveni stupanj, spajanje je završeno. Pokazivač minimuma se ažurira i postavlja na korijen s najmanjom vrijednosti, prikazujući konačni izgled hrpe.

Operacija *extract min* ima vremensku složenost $O(log\ n)$, (19).



Slika 18. Koraci operacije extract min, *(19)*

4.2.5 Operacija Decrease Key

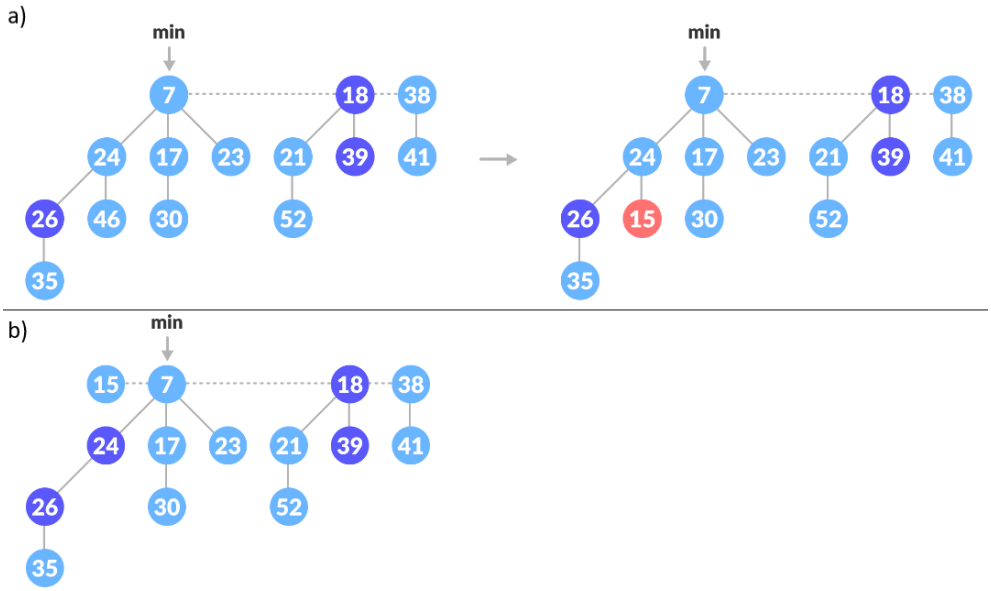
Ažuriranje težine vrha također je važan proces koji se izvodi tijekom Dijkstrinog algoritma. Ovaj zadatak izvršava operacija *decrease key*.

Operacija *decrease key* smanjuje vrijednost određenog čvora i prilagođava strukturu hrpe kako bi se održala njena svojstva. Da bi se u potpunosti razumjela njena funkcionalnost, operacija je objašnjena kroz dva primjera, (21).

Na **Slika 19** prikazani su koraci prvog primjera, koji uključuju sljedeće:

a) Vrijednost čvora 46 smanjuje se na 15. Treba obratiti pažnju na tamno plave čvorove, što označava da su već izgubili jedno dijete. Kada čvor izgubi drugo dijete, on se reže i dodaje u popis korijena kako bi se očuvala struktura i svojstva hrpe.

b) Budući da je čvor 15 manji od svog roditelja, čvora 24, on se reže i dodaje u popis korijena, dok se čvor 24 označava tamno plavom bojom zbog gubitka jednog djeteta.

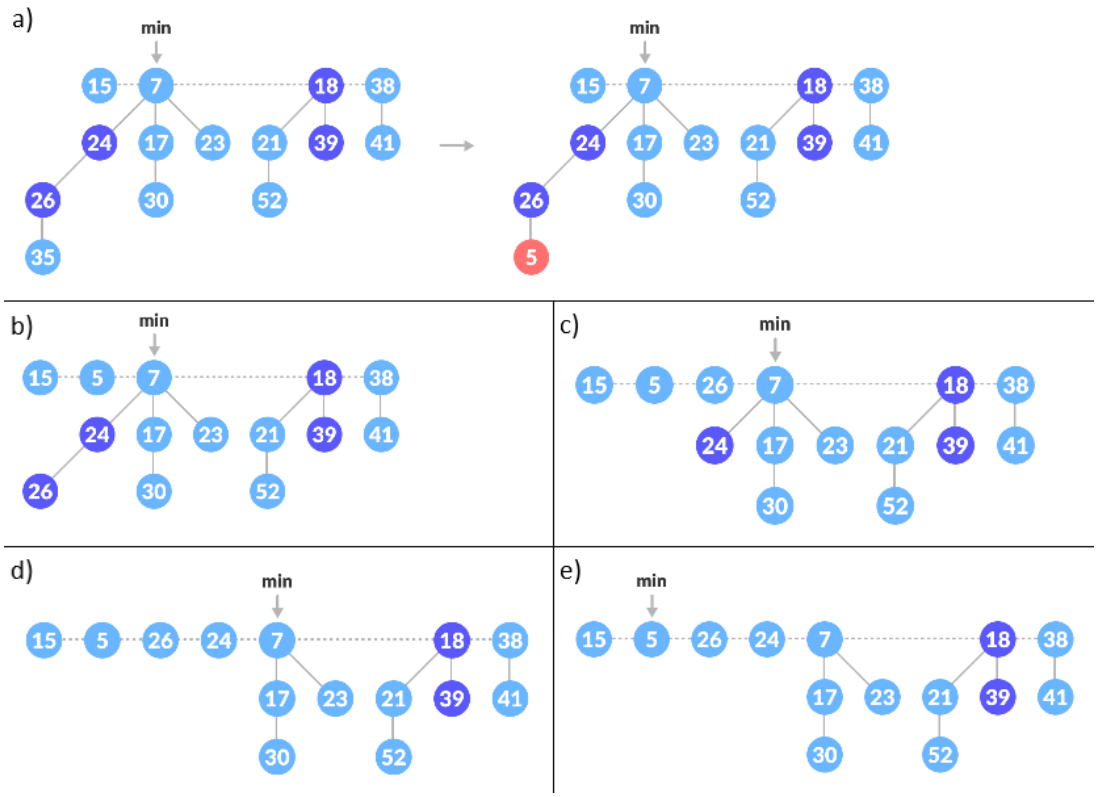


Slika 19. Operacija decrease key - prvi primjer, *(21)*

Na **Slika 20** prikazani su koraci drugog primjera, koji uključuju sljedeće:

a) Vrijednost čvora 35 smanjuje se na 5.
b) Čvor 5 je manji od svog roditelja 26, pa se izrezuje i dodaje u popis korijena. Čvor 26 se označava tamno plavom bojom jer je izgubio drugo dijete (u prethodnom primjeru bio je označen nakon gubitka prvog djeteta).
c) Čvor 26 se izrezuje i dodaje u popis korijena jer je izgubio oba djeteta, dok se čvor 24 označava tamno plavom bojom jer je izgubio drugo dijete (u prethodnom primjeru je izgubio jedno dijete).
d) Čvor 24 se izrezuje i dodaje u popis korijena jer je izgubio oba djeteta.
e) Pokazivač se ažurira i postavlja na korijen s najmanjom vrijednosti. Završni izgled hrpe je prikazan. Svi čvorovi koji su izgubili oba djeteta označeni su svijetlo plavom bojom, što znači da više nemaju djece.

Operacija *decrease key* ima vremensku složenost $O(1)$, (21).



Slika 20. Operacija decrease key - drugi primjer, *(21)*

4.3 Rezultati implementacije Fibonaccijeve hrpe

U svrhu usporedbe Dijkstra algoritma s Fibonaccijevom hrpom i bez nje, provedena je simulacija u C# programskom jeziku. Simulacijom su dobivena vremena izvršavanja algoritama, a obuhvaćeno je ukupno 2206 testiranja, raspoređenih u četiri kategorije:

- 1000 simulacija za udaljenosti do 10 kilometara
- 1000 simulacija za udaljenosti između 10 i 50 kilometara
- 185 simulacija za udaljenosti između 50 i 100 kilometara
- 21 simulacija za udaljenosti između 100 i 600 kilometara

Cilj simulacije bio je analizirati performanse algoritama. Za svaki test izračunati su prosječno vrijeme izvršavanja, standardna devijacija, minimalno i maksimalno vrijeme izvršavanja. Algoritmi su implementirani kao jednodretveni proces, bez paralelizacije, što je omogućilo objektivno mjerenje svakog algoritma bez utjecaja paralelnih opcija. Simulacija je provedena na računalu s 32 GB RAM-a i AMD-ovim 4.2 GHz Threadripper procesorom.

Rezultati za udaljenosti do 10 kilometara prikazani su u **Tablica 5**. Prosječno vrijeme izvršavanja Dijkstra algoritma s Fibonaccijevom hrpom bilo je značajno manje (0,28 s) u usporedbi s algoritmom bez Fibonaccijeve hrpe (5,23 s). Standardna devijacija, koja ukazuje na varijabilnost vremena izvršavanja, također je bila mnogo manja kod algoritma s Fibonaccijevom hrpom (0,8 s), što upućuje na dosljedniju izvedbu. Najmanje vrijeme izvršavanja kod Dijkstre s Fibonaccijevom hrpom iznosilo je 0,19 sekundi, dok je maksimalno vrijeme izvršavanja kod Dijkstra algoritma bez Fibonaccijeve hrpe iznosilo 199,65 sekundi. Minimalne vrijednosti su slične, ali su maksimalne vrijednosti kod algoritma bez Fibonaccijeve hrpe znatno veće.

Tablica 5. Rezultati simulacija do 10 kilometara

	Prosjek [s]	St. dev [s]	Max vrijeme [s]	Min vrijeme [s]
Dijkstra sa Fibonaccijevom hrpom	0,28	5,23	0.56	0,19
Dijkstra bez Fibonaccijeve hrpe	5,23	12,81	199,65	0

Slični trendovi opaženi su i za udaljenosti između 10 i 50 kilometara, kako je prikazano u **Tablica 6**. Prosječno vrijeme izvršavanja Dijkstra algoritma s Fibonaccijevom hrpom iznosilo je 0,39 sekundi, dok je prosječno vrijeme za Dijkstra algoritam bez Fibonaccijeve hrpe znatno duže, 388,74 sekunde. Standardna devijacija bila je manja kod Dijkstra algoritma s Fibonaccijevom hrpom (0,12 s) u usporedbi s algoritmom bez Fibonaccijeve hrpe (580,07 s). Maksimalno vrijeme izvršavanja za Dijkstra algoritam bez Fibonaccijeve hrpe moglo je doseći do 3838,88 sekundi, dok je najveće vrijeme za Dijkstra algoritam s Fibonaccijevom hrpom iznosilo samo 0,87 sekundi.

Tablica 6. Rezultati simulacija između 10 i 50 kilometara

	Prosjek [s]	St. dev [s]	Max vrijeme [s]	Min vrijeme [s]
Dijkstra sa Fibonaccijevom hrpom	0,39	0,12	0,87	0,21
Dijkstra bez Fibonaccijeve hrpe	388,74	580,07	3838,88	0,07

Za udaljenosti između 50 i 100 kilometara, rezultati prikazani u **Tablica 7** pokazuju da je prosječno vrijeme izvršavanja Dijkstra algoritma s Fibonaccijevom hrpom bilo 0,55 sekundi, dok je prosječno vrijeme za algoritam bez Fibonaccijeve hrpe iznosilo 2240,49 sekundi. Standardna devijacija vremena također je bila manja kod Dijkstra algoritma s Fibonaccijevom hrpom (0,15 s) u usporedbi s algoritmom bez Fibonaccijeve hrpe (1774,53 s). Maksimalno vrijeme izvršavanja za algoritam bez Fibonaccijeve hrpe moglo je doseći 6892,40 sekundi, dok je najveće vrijeme kod Dijkstra algoritma s Fibonaccijevom hrpom iznosilo samo 1,03 sekundi. Minimalno vrijeme izvršavanja kod Dijkstra algoritma s Fibonaccijevom hrpom bilo je 0,28 sekundi, dok je minimalno vrijeme kod algoritma bez Fibonaccijeve hrpe bilo 19,15 sekundi.

Tablica 7. Rezultati simulacija između 50 i 100 kilometara

	Prosjek [s]	St. dev [s]	Max vrijeme [s]	Min vrijeme [s]
Dijkstra sa Fibonaccijevom hrpom	0,55	0,15	1,03	0,28
Dijkstra bez Fibonaccijeve hrpe	2240,49	1774,53	6892,40	19,15

Rezultati za udaljenosti između 100 i 600 kilometara, prikazani u **Tablica 8**, pokazali su da prosječno vrijeme za Dijkstra algoritam s Fibonaccijevom hrpom iznosi 1,03 sekundi, dok prosječno vrijeme za Dijkstra algoritam bez Fibonaccijeve hrpe prelazi 16175,04 sekundi. Standardna devijacija vremena također je znatno manja kod algoritma s Fibonaccijevom hrpom (0,27 s) u usporedbi s algoritmom bez Fibonaccijeve hrpe (9475,49 s). Maksimalno vrijeme izvršavanja za algoritam bez Fibonaccijeve hrpe moglo je doseći impresivnih 32223,33 sekundi, dok maksimalno vrijeme kod algoritma s Fibonaccijevom hrpom iznosi samo 1,63 sekundi. Minimalno vrijeme kod algoritma s Fibonaccijevom hrpom bilo je 0,52 sekundi, dok je minimalno vrijeme kod algoritma bez Fibonaccijeve hrpe bilo 1524,03 sekundi.

Tablica 8. Rezultati simulacija između 100 i 600 kilometara

	Prosjek [s]	St. dev [s]	Max vrijeme [s]	Min vrijeme [s]
Dijkstra sa Fibonaccijevom hrpom	1,03	0,27	1,63	0,52
Dijkstra bez Fibonaccijeve hrpe	16175,04	9475,49	32223,33	1524,03

Korištenje Fibonaccijeve hrpe u Dijkstra algoritmu značajno poboljšava učinkovitost, smanjujući prosječno vrijeme izvođenja i osiguravajući veću konzistentniju u izvedbi. Suprotno tome, Dijkstra algoritam bez Fibonaccijeve hrpe pokazuje veću varijabilnost i manje predvidljive performanse, što se manifestira kroz znatno duže prosječne i maksimalne vremenske periode izvršavanja. Ovi rezultati jasno naglašavaju prednosti primjene Fibonaccijeve hrpe u optimizaciji Dijkstra algoritma, ukazujući na značajnu prednost u smanjenju vremenskih odstupanja i poboljšanju ukupne učinkovitosti algoritma.

5 ODREĐIVANJE PROSTORNOG DOSEGA

Proces određivanja prostornog dosega uključuje nekoliko ključnih koraka. Prvo se identificira najbliži vrh za zadane koordinate. Nakon toga slijedi filtriranje ruta prema zadanim kriterijima i priprema koordinata za crtanje konture. Na kraju se generira kontura koja vizualno prikazuje prostorni doseg. Slijedi detaljan opis svakog koraka ovog procesa.

5.1 Određivanje najbližeg vrha i izračun udaljenosti

Prvi korak u određivanju prostornog dosega je identifikacija najbližeg vrha za zadane koordinate. U tu svrhu koristi se metoda *PronadiNajbliziLink*, prikazana na **Slika 21**. Ovoj metodi predaju se izabrane koordinate, tj. geografska dužina i širina. Metoda započinje stvaranjem varijable za novi vrh, označene kao *v3*, s praznim podacima te varijable *minUdaljenost* za pohranu minimalne udaljenosti. Nakon toga, za svaki vrh u listi vrhova računa se udaljenost do najbliže točke na liniji konture. Za dobivanje udaljenosti koristi se metoda *getDistanceFromPointToClosestPointOnLine*, kojoj se predaju koordinate početne i završne točke te geografska dužina i širina. Izračunata udaljenost uspoređuje se s varijablom *minUdaljenost*. Ako je nova izračunata udaljenost manja, ona postaje nova vrijednost *minUdaljenost*, a odgovarajući vrh postaje *v3*. Kada se metoda završi za svaki vrh u listi, dobiva se najbliži vrh.

```
1 reference
private Vrh PronadiNajbliziLink(double longitude, double latitude)
{
    Vrh v3 = null;
    double minUdaljenost = double.MaxValue;
    foreach (Vrh v4 in listaVrhova.Values)
    {
        double udaljenost = getDistanceFromPointToClosestPointOnLine(v4.xPocetak, v4.yPocetak, v4.xZavrsetak, v4.yZavrsetak, longitude, latitude);
        if (udaljenost < minUdaljenost)
        {
            minUdaljenost = udaljenost;
            v3 = v4;
        }
    }
    return v3;
}
```

Slika 21. Kod metode za pronalazak najbližeg vrha

Za izračunavanje vrijednosti varijable udaljenosti koristi se metoda *getDistanceFromPointToClosestPointOnLine*, prikazana na **Slika 22**. Ova metoda iz predanih varijabli stvara dva vektora. Prvi vektor, *vec_l1P*, predstavlja udaljenost između početne točke

i koordinata geografskih dužina i širina, dok drugi vektor, *vec_l1l2*, predstavlja udaljenost između početne i završne točke. Kako bi se odredila udaljenost točke od najbliže točke na liniji, potrebno je prvo izračunati Euklidsku udaljenost te skalarni produkt prethodno navedenih vektora. Iz ovih izračuna dobiva se najbliža početna točka. Zatim se pomoću metode *airalDistHaversine* izračunava zračna udaljenost.

```
1 reference
public static double getDistanceFromPointToClosestPointOnLine(double lx1, double ly1, double lx2, double ly2, double px, double py)
{

    double[] vec_l1P = new double[2] { px - lx1, py - ly1 };
    double[] vec_l1l2 = new double[2] { lx2 - lx1, ly2 - ly1 };

    double mag = Math.Pow(vec_l1l2[0], 2) + Math.Pow(vec_l1l2[1], 2);
    double prod = vec_l1P[0] * vec_l1l2[0] + vec_l1P[1] * vec_l1l2[1];
    double normDist = prod / mag;

    double clX = lx1 + vec_l1l2[0] * normDist;
    double clY = ly1 + vec_l1l2[1] * normDist;


    double minLX = Math.Min(lx1, lx2);
    double minLY = Math.Min(ly1, ly2);
    double maxLX = Math.Max(lx1, lx2);
    double maxLY = Math.Max(ly1, ly2);
    if (clX < minLX)
    {
        clX = minLX;
    }
    if (clY < minLY)
    {
        clY = minLY;
    }

    if (clX > maxLX)
    {
        clX = maxLX;
    }
    if (clY > maxLY)
    {
        clY = maxLY;
    }

    return airalDistHaversine(px, py, clX, clY);
}
```

Slika 22. Kod metode za dobivanje udaljenosti točke od najbliže točke na liniji

Za izračun zračne udaljenosti koristi se metoda *airalDistHaversine*, prikazana na **Slika 23**. Ova metoda uzima geografsku širinu i dužinu te koordinate točke. Za izračun zračne udaljenosti odabrane lokacije i najbliže točke na sferi koristi se Haversine-ova formula. Formula uzima u obzir radijus Zemlje, kao i vrijednosti φ (geografska širina) i λ (geografska dužina). Potrebno je prvo izračunati $\varphi 1$ i $\varphi 2$, gdje se φ izračunava množenjem geografske širine točke s π i dijeljenjem s 180. λ se izračunava množenjem geografske dužine s π i dijeljenjem s 180. Nakon izračuna zračne udaljenosti, dobivena vrijednost vraća se prethodnoj metodi.

```
1 reference
public static double airalDistHaversine(double lon1, double lat1, double lon2, double lat2)
{

    double R = 6371000;
    double phi1 = lat1 * Math.PI / 180;
    double phi2 = lat2 * Math.PI / 180;
    double deltaphi = (lat2 - lat1) * Math.PI / 180;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180;

    double a = Math.Sin(deltaphi / 2) * Math.Sin(deltaphi / 2) +
               Math.Cos(phi1) * Math.Cos(phi2) *
               Math.Sin(deltaLambda / 2) * Math.Sin(deltaLambda / 2);
    double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));

    double d = R * c;
    return d;
}
```

Slika 23. Kod metode za računanje zračne udaljenosti dvije točke na sferi

5.2 Filtriranje i priprema koordinata

Nakon pokretanja Dijkstrinog algoritma, koji je detaljno objašnjen u ranijem poglavlju, dobivene rute se filtriraju na temelju kriterija ograničenja, kao što su vrijeme ili energetska potrošnja. Rute koje zadovoljavaju ova ograničenja zadržavaju se za daljnju obradu, kao što je prikazano na **Slika 24**.

```
//Filtiranje vrhova
var listaVrhovaZaFilter = listaVrhova.ToList();
//NE tezina nego
listaVrhovaZaFilter=listaVrhovaZaFilter.FindAll(vrh => vrh.Value.tezina < n);
listaVrhovaZaFilter = listaVrhovaZaFilter.FindAll(vrh => vrh.Value.tezina > 0);
```

Slika 24. Kod za filtriranje ruta na temelju kriterija ograničenja

Prije nego se pristupi stvaranju konture, potrebno je prikupiti sve relevantne koordinate koje će biti korištene za crtanje. Koordinate su pripremljene iz filtriranih ruta na način da se generiraju točke duž svake rute. Ovisno o duljini rute, mogu se dodati dodatne točke između početne i završne točke rute kako bi se dobila točnija reprezentacija. Ako je ruta duža od 50 metara, generiraju se intervalne točke, dok se za kraće rute dodaju samo početne, završne i srednje točke, kao što je prikazano na **Slika 25**.

```
// Create a list of coordinates from the filtered data
var coordinates = new List<Coordinate>();
foreach (KeyValuePair<int,Vrh> kvPair in listaVrhovaZaFilter)
{
    Vrh tlink = kvPair.Value;
    int duljinaL = tlink.duljinaLinka;
    int numIntervals = (int)(duljinaL / 50);

    if (duljinaL > 50)
    {
        for (int i = 0; i <= numIntervals; i++)
        {
            double ratio = (double)i / numIntervals;
            double newLat = tlink.yPocetak + ratio * (tlink.yZavrsetak - tlink.yPocetak);
            double newLon = tlink.xPocetak + ratio * (tlink.xZavrsetak - tlink.xPocetak);
            var coord = new Coordinate(newLon, newLat);
            coordinates.Add(coord);
        }
    }
    else
    {
        var coord = new Coordinate(kvPair.Value.xZavrsetak, kvPair.Value.yZavrsetak);
        coordinates.Add(coord);
        coord = new Coordinate(kvPair.Value.xPocetak, kvPair.Value.yPocetak);
        coordinates.Add(coord);
        coord = new Coordinate((kvPair.Value.xPocetak + kvPair.Value.xZavrsetak) / 2, (kvPair.Value.yPocetak + kvPair.Value.yZavrsetak) / 2);
        coordinates.Add(coord);
    }
}
```

Slika 25. Kod za stvaranje koordinata iz zadržanih ruta

5.3 Generiranje konture

Nakon što su koordinate pripremljene, koriste se za stvaranje geometrijskih objekata. *GeometryFactory* se koristi za kreiranje *MultiPoint* objekta iz prikupljenih koordinata, kao što je prikazano na **Slika 26**. Ovaj objekt predstavlja skup svih točaka koje će biti korištene za izračunavanje konture.

```
// Create a geometry from the coordinates
GeometryFactory geomFactory = new GeometryFactory();
MultiPoint geometry = geomFactory.CreateMultiPointFromCoords(coordinates.ToArray());
```

Slika 26. Kod za stvaranje geometrijskih objekata

Na temelju geometrijskog objekta stvorenog iz koordinata, generira se kontura koristeći metode *ConcaveHullv2* ili *ConvexHull*, prikazane na **Slika 27**. Konkavna kontura bolje prati raspored točaka, dok konveksna kontura predstavlja najjednostavniji oblik koji obuhvaća sve točke. Razlika između ovih metoda leži u vrijednosti varijable *alpha shape*. Povećanjem vrijednosti *alpha shape*, graf postaje konveksan. Konkavni grafovi zauzimaju manju površinu i zbog toga su prikladniji za opis realnih problema, kao što su određivanje najkraćih ruta prema vremenu putovanja ili energetske potrošnje.

```
1 reference
private static Geometry ConcaveHullv2(Geometry input)
{
    return ConcaveHull.ConcaveHullByLength(input, 0.0015);
}

1 reference
private static Geometry ConvexHull(Geometry input)
{
    var buffer = input.ConvexHull();
    return buffer;
}
```

Slika 27. Kod metoda za crtanje konkavnih i konveksnih grafova

Rezultat konture se zatim koristi za prikazivanje na karti, omogućujući vizualni prikaz prostornog dosega za zadane kriterije. Kako bi se konture prikazale, potrebno je grafičko sučelje čija izrada detaljno slijedi u sljedećem poglavlju.

6 GRAFIČKO SUČELJE

Aplikacija za vizualizaciju najkraćih ruta, vremena putovanja i energetske potražnje izrađena je u ASP.NET okviru koristeći Model-View-Controller (MVC) obrazac. MVC obrazac sastoji se od tri ključne komponente: modela (engl. Model), pogleda (engl. View) i kontrolera (engl. Controller). Ova podjela omogućuje jasno razdvajanje informacija od načina na koji se te informacije prezentiraju korisniku. Također, MVC omogućuje odvajanje poslovne logike, logike korisničkog sučelja i ulazne logike, čime se osigurava bolja kontrola nad HTML-om i URL-ovima, što znatno olakšava proces razvoja web aplikacija, (22).

Model predstavlja cjelokupnu logiku podataka koju koriste korisnici. Njegova glavna zadaća je prenošenje podataka između pogleda i kontrolera. Model se bavi pristupom, pohranom i manipulacijom podacima, osiguravajući da su podaci ispravno preneseni i obrađeni unutar sustava, (22).

Kontroler omogućuje „komunikaciju“ između modela i pogleda. Njegova zadaća je slanje naredbi modelu radi manipulacije podacima te prosljeđivanje tih manipuliranih podataka pogledu. Kontroler prima korisničke ulaze, obrađuje ih i usmjerava odgovarajuće akcije prema modelu, čime se osigurava dinamična interakcija unutar aplikacije, (22).

Pogled predstavlja korisničko sučelje aplikacije. On generira korisničko sučelje koje se prikazuje korisniku te prikazuje manipulirane podatke koje je dobio od kontrolera. Pogled je odgovoran za prezentaciju podataka i omogućuje korisnicima interakciju s aplikacijom putem intuitivnog i preglednog sučelja, (22).

Primjer rada MVC obrasca može se demonstrirati kroz scenarij u kojem krajnji korisnik želi dobiti popis cestovnih segmenata (linkova ceste). Proces započinje kada korisnik pošalje zahtjev poslužitelju, koji taj zahtjev prosljeđuje odgovarajućem kontroleru. Kontroler zatim upućuje modelu zahtjev za dohvaćanjem podataka o cestovnim segmentima. Model pristupa bazi podataka kako bi pronašao relevantne informacije i vraća popis svih cestovnih segmenata kontroleru. Nakon što kontroler primi popis, prosljeđuje ga pogledu. Pogled tada generira korisničko sučelje koje prikazuje popis cestovnih segmenata krajnjem korisniku.

6.1 Struktura web aplikacija

Sama web aplikacija se sastoji od dva dijela: frontend-a i backend-a.

Frontend je dio web aplikacije koji korisnik direktno vidi i koristi. Ovaj segment aplikacije sastoji se od pogleda, koji kreiraju korisničko sučelje. Pogledi su odgovorni za prikazivanje podataka korisnicima na pregledan i intuitivan način, omogućujući im interakciju s aplikacijom.

Backend je dio web aplikacije koji krajnji korisnik ne vidi niti direktno koristi. Ovaj segment predstavlja serversku stranu aplikacije koja pohranjuje, upravlja i distribuira podatke, te osigurava efikasan rad same aplikacije. Backend obuhvaća kontroler i model.

Za lakšu izradu web aplikacije korišteni su NuGet paketi. NuGet je službeni upravitelj paketa za .NET, koji omogućuje programerima da lako preuzimaju i koriste različite softverske biblioteke i dijelove koda u svojim projektima. Ovi paketi sadrže kompilirani kod zajedno sa svim potrebnim metapodacima, čime olakšavaju integraciju i upravljanje ovisnostima unutar .NET aplikacija, (23).

U ovoj web aplikaciji korišteni su sljedeći NuGet paketi:

- Bootstrap
- FibonacciHeap
- jQuery
- Microsoft.AspNet.MVC
- Newtonsoft.Json

Za razvoj frontenda ove web aplikacije korišten je Bootstrap. Bootstrap omogućuje brzu i efikasnu izradu prilagodljivih i modernih korisničkih sučelja. Pruža bogat set CSS i JavaScript komponenti koje se lako integriraju u aplikaciju, olakšavajući stvaranje atraktivnog i funkcionalnog sučelja.

FibonacciHeap paket omogućuje korištenje algoritma Fibonaccijeve hrpe. Ovaj paket pruža unaprijed napisane metode za sve operacije koje Fibonaccijeva hrpa podržava, čime se eliminira potreba za dodatnim pisanjem složenog koda.

jQuery je JavaScript biblioteka koja pojednostavljuje manipulaciju elementima na web stranici, upravljanje događajima i komunikaciju s poslužiteljem. Korištena je za izradu interaktivnog i dinamičkog frontenda, omogućujući jednostavniji i učinkovitiji razvoj.

Microsoft.AspNet.MVC je paket koji podržava izradu dinamičkih web aplikacija koristeći MVC arhitekturu.

Newtonsoft.Json je biblioteka za rad s JSON podacima. Omogućuje lako pretvaranje podataka u JSON format i obrnuto, što je ključno za moderne web aplikacije koje često ovise o razmjeni podataka u JSON formatu između frontenda i backenda.

6.1.1 Backend web aplikacije

Backend aplikacije obuhvaća ključne algoritme koji su detaljno obrađeni u prethodnim poglavljima. U ovaj dio spadaju postupci učitavanja podataka, Dijkstrin i Bellman-Fordov algoritam, te implementacija Fibonaccijeve hrpe kao strukture podataka. Ovi algoritmi i strukture čine temelj serverske strane aplikacije, omogućujući efikasnu obradu i upravljanje podacima, što osigurava optimalno funkcioniranje i pouzdanost cijele aplikacije.

6.1.2 Frontend web aplikacije

Frontend web aplikacije obuhvaća sve korisničke sučelja i poglede. Aplikacija se sastoji od dva glavna pogleda: *Home* i *Map*, implementiranih korištenjem HTML-a i JavaScript-a.

Pogled koji strukturira aplikaciju na dvije stranice naziva se *_Layout.cshtml*. U njemu se nalazi kod koji definira navigacijsku traku s dva gumba koji omogućuju pristup stranicama *Home* i *Map*. Na **Slika 28** prikazan je kod koji implementira ovu navigacijsku traku unutar *_Layout.cshtml* pogleda.

```
<nav class="navbar fixed-top navbar-expand-lg navbar-dark bg-black align-items-center">
    <div class="container">
        @Html.ActionLink("Student project", "Index", "Home", new { area = "" }, new { @class = "navbar-brand custom-navbar-item" })
        <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" title="Toggle navigation" aria-controls="navbarSupportedContent"
                aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse d-sm-inline-flex justify-content-between">
            <ul class="navbar-nav flex-grow-1">
                <li>@Html.ActionLink("Home", "Index", "Home", new { area = "" }, new { @class = "nav-link custom-navbar-item" })</li>
                <li>@Html.ActionLink("Map", "Index", "Map", new { area = "" }, new { @class = "nav-link custom-navbar-item" })</li>
            </ul>
        </div>
    </div>
</nav>
```

Slika 28. Kod za stvaranje navigacijske trake web aplikacije

Također, *_Layout.cshtml* sadrži kod koji definira podnožje web aplikacije. Podnožje uključuje informacije o autorima aplikacije i mentorima. Kod za kreiranje podnožja web aplikacije može se vidjeti na **Slika 29**.

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year Student project: Dumančić, Knez i Kukor (mentors: prof. Tonči Carić, PhD, Tomislav Erdelić, PhD) </p>
    </footer>
</div>
```

Slika 29. Kod za kreiranje podnožja web aplikacije

Stranica *Home*, prikazana na **Slika 30**, sadrži naslov rada, imena autora i mentora, te kratki opis aplikacije na engleskom jeziku. Također, na *Home* stranici se nalazi primjer iscrtanog konkavnog prostornog poligona dosega označen crvenom bojom koji prikazuje udaljenost kojom osoba može putovati automobilom u roku od odabranih pet minuta od lokacije čije su koordinate: 45.808820054643576, 15.974979400634764. Kod koji definira izgled stranice *Home* može se pronaći u pogledu *Home* pod nazivom *Indeks.cshtml*. Za stilizaciju teksta na stranici *Home* koristi se sljedeći HTML kod, prikazan na **Slika 31**.

STUDENT PROJECT HOME MAP

PROJECT TITLE:

Visualization of the shortest travel time and energy consumption routes on a road network

Mentor: prof. Tonči Carić, PhD

Supervisor: Tomislav Erdelić, PhD

Creators:

Ante Dumančić

Dominik Knez

Marko Kukor

Department: Department of Intelligent Transportation System

Call: Internal competition of the Faculty of Transport and Traffic Sciences for the implementation of research projects by undergraduate and graduate students.

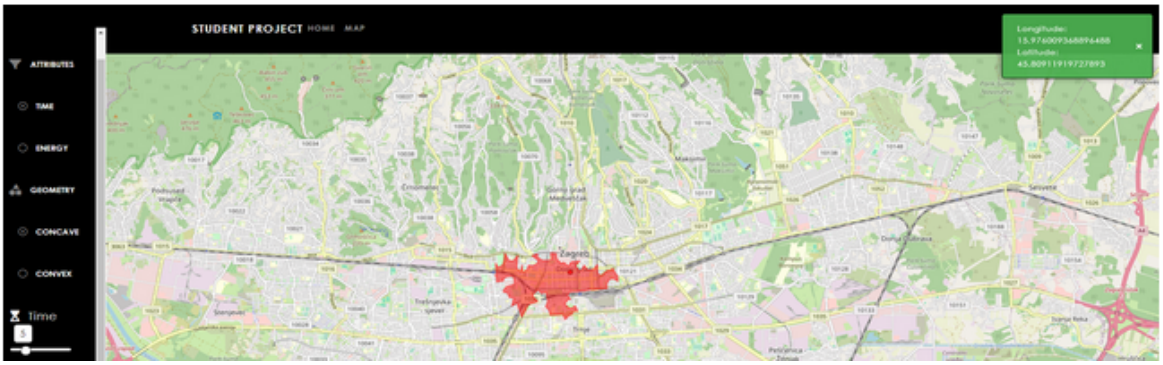
Funding: Faculty of Transport and Traffic Sciences

Budget: 1650 EUR

Project description:

The project aims to develop a web application that analyzes routes based on travel time or energy consumption criteria. Utilizing a modeled representation of the actual road network, the application generates a graph derived from a digital traffic map obtained through the SORDITO project. Within this graph, speed profiles developed as part of the SORDITO project are employed to compute travel time. For determining energy consumption, these speed profiles are integrated with the Copernicus digital elevation model and Nissan Leaf vehicle characteristics within the vehicle longitudinal dynamic model. The application incorporates the implementation of Dijkstra's and Bellman-Ford's algorithms, crucial for identifying the shortest routes in terms of travel time and energy consumption on the road network. To enhance algorithm execution, a Fibonacci heap data structure is applied. Spatial concave and convex functions were employed to delineate the geographical region accessible within a specified travel time or energy threshold. Featuring an interactive graphical interface, the application facilitates spatial visualization using contours. Users can dynamically alter the initial point, contour type, as well as time and energy parameters through the graphical web interface. Presently, the geographical scope is confined to the City of Zagreb.

Example below shows the concave countour (red) that a person can achieve by car within 5 minutes starting from the position with coordinates: 45.808820054643576, 15.974979400634764.



Slika 30. Prikaz stranice Home web aplikacije

```
<style>
    .lead {
        font-family: 'Arial', sans-serif;
        line-height: 1.6;
        color: #333;
    }

        .lead h2 {
            color: #333;
            font-size: 24px;
            margin-bottom: 10px;
        }

        .lead p, .lead li {
            margin-bottom: 10px;
        }

        .lead .highlight {
            font-weight: bold;
            color: #555;
        }

        .lead .creator-list {
            list-style: none;
            margin: 0;
            padding: 0;
        }

            .lead .creator-list li {
                display: block;
                margin-bottom: 5px;
            }

        .lead .department {
            font-style: italic;
            color: #444;
        }

        .lead .project-description {
            font-size: 18px;
            text-align: justify;
        }
</style>
```

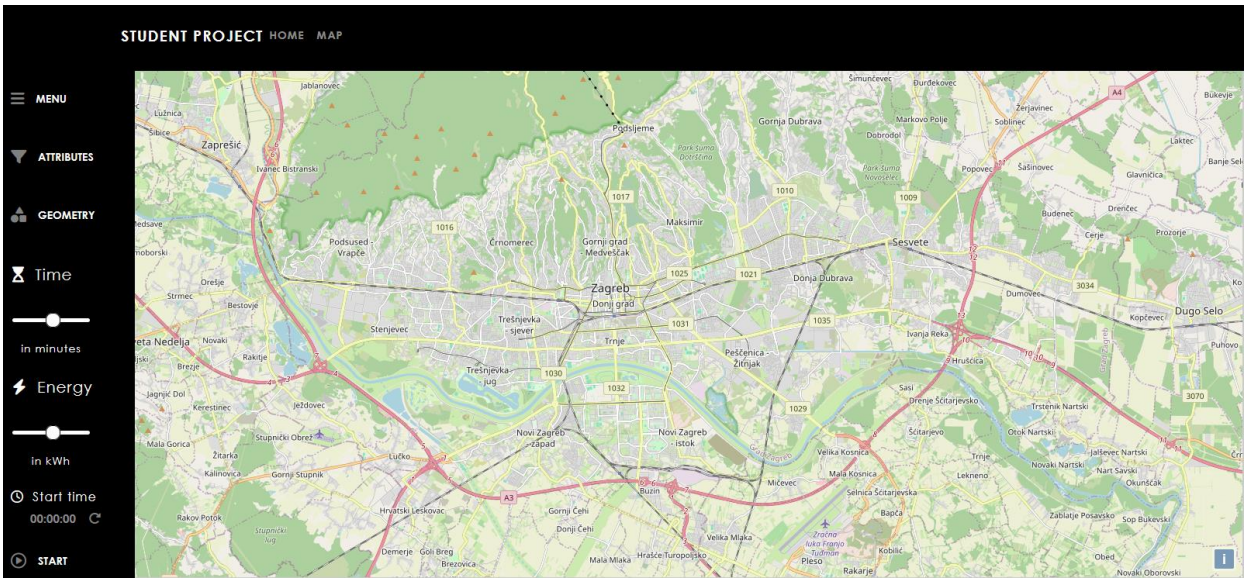
Slika 31. HTML kod za dizajn teksta na stranici *Home*

Za dodavanje slike koja prikazuje primjer iscrtanog grafa koristio se kod prikazan na **Slika 32**.

```
<img src="~/Content/Images/SP_example.png"
     alt="This is a picture of example"
     height="600">
```

Slika 32. Kod za dodavanje slike na stranicu *Home*

Stranica *Map* sadrži pozadinsku *OpenStreetMap* (OSM) kartu i bočnu traku s parametrima koje korisnik može odabrati za crtanje grafa. Korisničko sučelje ove stranice prikazano je na **Slika 33**.



Slika 33. Prikaz stranice *Map* web aplikacije

Pogled *Map* je podijeljen na dva dijela: prvi dio sadrži kod za dizajn i funkcionalnost bočne trake, dok se drugi dio odnosi na kod za dizajn karte i funkcionalnost crtanja grafova na karti na temelju odabranih parametara s bočne trake.

Bočna traka sadrži pet vrsta filtara i uključuje opciju za proširivanje okvira trake pritiskom na gumb *Menu*. Dodatno, pritiskom na gumbove *Attributes* i *Geometry* proširuju se okviri kako bi omogućili odabir dodatnih opcija parametara. Kod za implementaciju mogućnosti proširivanja okvira prikazan je na **Slika 34**.

```
function togglePanel(panelId) {
    const panel = document.getElementById(panelId);

    if (panel.style.display === 'none' || panel.style.display === '') {
        panel.style.display = 'block';

        if (panelId === 'panel1' && panel.childElementCount === 0) {
            initializePanel1();
        } else if (panelId === 'panel2' && panel.childElementCount === 0) {
            initializePanel2();
        }
    } else {
        panel.style.display = 'none';
    }
}
```

Slika 34. Kod za proširivanje okvira bočne trake

Prilikom proširivanja okvira u prethodno spomenutom kodu, primjenjuju se funkcije koje inicijaliziraju opcije za Atribute i Geometrije. Na **Slika 35** prikazan je kod koji prikazuje kako se inicijaliziraju opcije parametara.

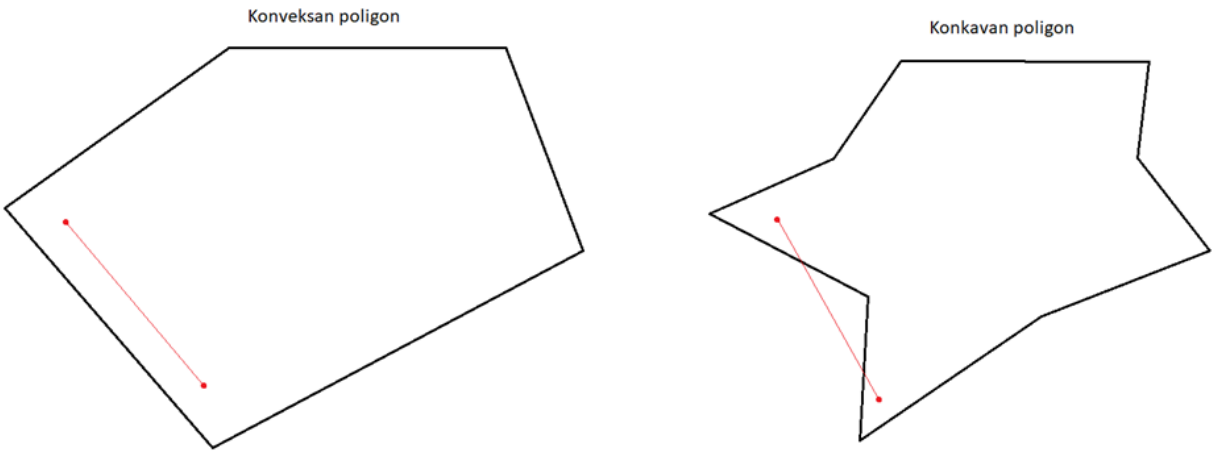
```
function initializePanel1() {
    const panelButton1 = createPanelButton('Time', 'side-button', 'fa-regular fa-circle', 'fa-regular fa-circle-xmark', 'panelButton1', 'panelButton2', 'time', 'energy');
    const panelButton2 = createPanelButton('Energy', 'side-button', 'fa-regular fa-circle', 'fa-regular fa-circle-xmark', 'panelButton2', 'panelButton1', 'energy', 'time');

    document.getElementById('panel1').appendChild(panelButton1);
    document.getElementById('panel1').appendChild(panelButton2);

    restoreSelectedState(panelButton1);
    restoreSelectedState(panelButton2);
}

function initializePanel2() {
    const panelButton3 = createPanelButton('Concave', 'side-button', 'fa-regular fa-circle', 'fa-regular fa-circle-xmark', 'panelButton3', 'panelButton4', 'concave', 'convex');
    const panelButton4 = createPanelButton('Convex', 'side-button', 'fa-regular fa-circle', 'fa-regular fa-circle-xmark', 'panelButton4', 'panelButton3', 'convex', 'concave');

    document.getElementById('panel2').appendChild(panelButton3);
    document.getElementById('panel2').appendChild(panelButton4);

    restoreSelectedState(panelButton3);
    restoreSelectedState(panelButton4);
}
```

Slika 35. Kod za inicijalizaciju opcija parametara Atributa i Geometrije.

Atributi u kontekstu crtanja grafova na karti odnose se na parametre prema kojima se odabiru linkovi za izračun najkraće rute od odabrane lokacije. Opcije parametra uključuju vrijeme i energiju.

Geometrija se odnosi na parametar koji definira vrstu grafa za najkraću rutu. Grafovi mogu biti konkavni ili konveksni. Konveksni grafovi su oni u kojima su svi pravci koji povezuju dvije točke unutar grafa potpuno unutar područja grafa. S druge strane, konkavni grafovi su oni u kojima se barem jedan pravac koji povezuje dvije točke izvan područja grafa.

Razlika između konkavnih i konveksnih grafova prikazana je na **Slika 36**, gdje je jasno istaknuta geometrijska karakteristika koja razlikuje ove dvije vrste grafova.



Slika 36. Razlika konveksnih i konkavnih grafova.

Prilikom inicijalizacije opcija parametara, stvaraju se tipke za odabir parametara koje korisnik može izabrati. Nakon što korisnik odabere željeni parametar, ta vrijednost se sprema u varijable koje se šalju kontroleru kako bi mogao generirati listu linkova. Kod za stvaranje tipki parametara i spremanje odabranih parametara u varijable može se vidjeti na **Slika 37**.

```
function createPanelButton(text, className, defaultIconClass, toggleIconClass, buttonId, otherButtonId, type, otherType) {
    const panelButton = document.createElement('button');
    panelButton.className = `btn btn-lg btn-primary ${className}`;
    panelButton.type = 'button';
    panelButton.id = buttonId;

    const defaultIconElement = document.createElement('i');
    defaultIconElement.className = defaultIconClass;
    defaultIconElement.style.fontSize = '15px';

    panelButton.appendChild(defaultIconElement);

    panelButton.innerHTML += ` ${text}`;

    panelButton.addEventListener('click', function () {
        console.log(`${text} clicked`);

        const otherButton = document.getElementById(otherButtonId);

        panelButton.firstChild.className = toggleIconClass;
        otherButton.firstChild.className = defaultIconClass;

        if (type === 'concave' || type === 'convex') {
            KontureType = `${text} Hull`;
            console.log(`KontureType changed to: ${KontureType}`);
            console.log(`Selected Type: ${selectedType}`);
        }

        if (type === 'time' || type === 'energy') {
            selectedType = type;
            console.log(`KontureType changed to: ${KontureType}`);
            console.log(`Selected Type: ${selectedType}`);
        }

    });

    return panelButton;
}
```

Slika 37. Kod za kreiranje tipka za odabir parametara

Nakon što korisnik odabere parametre Atributa, vrijeme ili energiju, potrebno je izabrati konkretne vrijednosti vremena ili energije. Za odabir ovih vrijednosti koristi se klizač (engl. Slider). Vrijednosti parametra vremena izražene su u minutama, dok su vrijednosti energije izražene u kilovat-satima (kWh). Kod za kreiranje klizača prikazano je na **Slika 38**.

```
function updateValue(elementId, value, labelId) {
    const slider = document.getElementById(elementId.replace('value', 'customRange'));
    const sliderValueElement = document.getElementById(elementId);
    const percent = (slider.value - slider.min) / (slider.max - slider.min);
    const offsetLeft = percent * slider.offsetWidth;
    sliderValueElement.style.left = slider.offsetLeft + offsetLeft + 'px';

    const formattedValue = (elementId === 'value2') ? value.replace('.', ',') : value;

    sliderValueElement.innerText = formattedValue;

    sliderValueElement.style.display = 'block';

    const hiddenInput = document.getElementById('hidden' + elementId.charAt(elementId.length - 1));
    hiddenInput.value = formattedValue;

    submitForm();
}
```

Slika 38. Kod za kreiranje klizača za odabir vrijednosti vremena ili energije

Nakon što korisnik odabere parametre Atributa i Geometrije, važno je omogućiti unos vremena početka putovanja. Ovo je ključno za simuliranje prometnih uvjeta u stvarnom okruženju, posebno uzimajući u obzir vršne sate kada je prometno opterećenje obično veće.

Za unos vremena početka koristi se okvir gdje korisnik može unijeti željeno vrijeme. Također, uključena je opcija za resetiranje vremena početka na početnu vrijednost, postavljenu na ponoć (00:00:00), kako bi se omogućilo ponovno postavljanje vremena ako je potrebno. Ovaj dio funkcionalnosti, uključujući okvir za unos početnog vremena i tipku za ponovno postavljanje, prikazan je na **Slika 39**.

```
const timeInput = document.getElementById("timeInput");

timeInput.addEventListener("input", (e) => {
    const valuesOfInput = e.target.value.replace(/[^0-9]/g, '').slice(0, 6);

    const formattedTime = valuesOfInput.replace(/(\d{2})(\d{0,2})(\d{0,2})/, (_, p1, p2, p3) => {
        let result = p1;
        if (p2 || p3) result += `:${p2 || '00'}${p3 ? `:${p3}` : ''}`;
        return result;
    });

    e.target.value = formattedTime;

});

function rotateTimeInput() {
    const timeInput = document.getElementById("timeInput");
    timeInput.value = "";
}
```

Slika 39. Kod za stvaranje okvira za unos vremena početka

Prilikom pokretanja stranice *Map*, inicijalno je postavljena početna točka na karti za Zagreb, s obzirom da aplikacija raspolaže podacima za taj grad. Kod koji postavlja početnu točku karte može se vidjeti na **Slika 40**.

```
var map = new ol.Map({
    interactions: ol.interaction.defaults({ doubleClickZoom: false }),
    target: 'divMap',
    layers: [
        new ol.layer.Tile({
            source: new ol.source.OSM()
        })
    ],
    view: new ol.View({
        center: ol.proj.fromLonLat([15.981179, 45.805281]),
        zoom: 12
    })
});
```

Slika 40. Kod za postavljanje početne točke karte

Za dodavanje početne lokacije radi kreiranja najkraće rute, korisnik mora dvostruko kliknuti na željenu lokaciju na karti. Nakon dvostrukog klika, otvara se prozorčić koji prikazuje koordinate odabrane lokacije. Te koordinate se zatim prosljeđuju funkciji za kreiranje markera na OSM karti. Kod za odabir lokacije na karti i prikaz koordinata odabrane lokacije može se vidjeti na **Slika 41**.

```
map.on('dblclick', function (event) {
    var koordinate = ol.proj.toLonLat(event.coordinate);

    longitude = koordinate[0];
    latitude = koordinate[1];

    document.getElementById('txtLatitude').value = latitude;
    document.getElementById('txtLongitude').value = longitude;

    var alertText = document.getElementById('alertText');
    alertText.textContent = 'Longitude: ' + longitude + ' Latitude: ' + latitude;

    var customAlert = document.getElementById('customAlert');
    customAlert.style.display = 'flex';

    setTimeout(function () {
        customAlert.style.display = 'none';
    }, 10000);

    addMarker([longitude, latitude]);
});
```

Slika 41. Kod za odabir lokacije na OSM karti i prikaz koordinata u prozorčiću

Za dodavanje markera na kartu na mjestu odabrane lokacije koristi se funkcija *addMarker*, koja prima koordinate odabrane lokacije dobivene iz prethodno opisane akcije. Marker na karti predstavljen je crvenim kružićem. Kod za dodavanje markera na kartu može se vidjeti na **Slika 42**.

```
function addMarker(coordinates) {
    var startButtonClicked = layerCreated;

    if (!startButtonClicked && markerFeature) {
        vectorSource.removeFeature(markerFeature);
        markerFeature = null;
    }

    markerFeature = new ol.Feature({
        geometry: new ol.geom.Point(ol.proj.fromLonLat(coordinates)),
    });

    var markerStyle = new ol.style.Style({
        image: new ol.style.Icon({
            src: 'https://upload.wikimedia.org/wikipedia/commons/e/ec/RedDot.svg',
            scale: 0.7,
        }),
    });

    markerFeature.setStyle(markerStyle);

    vectorSource.addFeature(markerFeature);

    layerCreated = false;
}
```

Slika 42. Kod za dodavanje markera za odabranu lokaciju na karti.

Aplikacija unutar funkcije koja kreira prostorne slike dosega najkraćih ruta uključuje funkciju upozorenja. Ta funkcija upozorava korisnika ako nije unio potrebne parametre kao što su koordinate lokacije, vrsta atributa i geometrija. Kada se upozorenje aktivira, otvara se tekstualni prozor koji obavještava korisnika o nedostajućim parametrima. Kod za funkciju upozorenja može se vidjeti na **Slika 43**.

```
message = "Error!"

if (longitude==0 && latitude==0) {
    message = message + "<br>Please choose coordinates!";
}
if (selectedType=="none") {
    message = message + "<br>Please choose attribute type!";
}
if (KontureType == "none") {
    message = message + "<br>Please choose geometry type!";
}

if (selectedType == "time" && $('#hidden1').val()==0) {
    message = message + "<br>Please choose time value!";
}
if (selectedType == "energy" && $('#hidden2').val() == 0) {
    message = message + "<br>Please choose energy value!";
}
if (sati>24||min>59||sek>59) {
    message = message + "<br>Please enter valid start time!";
}
if (message != "Error!") {
    var alarmText = document.getElementById('alarmText');
    alarmText.innerHTML = message;
    var customAlarm = document.getElementById('customAlarm');
    customAlarm.style.display = 'flex';
    setTimeout(function () {
        customAlarm.style.display = 'none';
    }, 10000);
}
```

Slika 43. Kod funkcije za upozoravanje

Funkcija *ComputeConture* je odgovorna za stvaranje grafa najkraće rute i u sebi pohranjuje varijable poput koordinata, početnog vremena putovanja, odabranih vrsta i vrijednosti atributa (energije ili vremena), kao i vrstu grafa (konkavan, konveksan). Nakon toga, te varijable se šalju kontroleru aplikacije. Za slanje ovih varijabli iz pogleda u kontroler koristi se *AJAX* poziv. *AJAX* poziv aktivira odgovarajuću metodu kontrolera koja obrađuje podatke. Kod za slanje varijabli prikazan je na **Slika 44**.

```
$.ajax({
    type: "POST",
    url: "@Url.Action("ComputeConture")",
    data: { longitude: longitude, latitude: latitude, selectedType: selectedType, selectedVarTime: $('#hidden1').val(), selectedVarEnergy: $('#hidden2').val(), KontureType: KontureType, strVrijemePocetka: strVrijemePocetka},
    success: succesfullCall,//Funkcija koja se poziva ako je poziv bio uspješan
    failure: function (response) {
    //Ispiši grešku u malom prozorčiću
    alert(response.error);
    }
});
```

Slika 44. Kod *AJAX* poziva

Ukoliko se metoda u kontroleru uspješno izvrši, pokreće se funkcija *successfulCall*, dok se u suprotnom slučaju prikazuje greška u prozorčiću. Funkcija *successfulCall* ima zadaću stvaranja sloja na OSM karti koji prikazuje prostorni doseg prema odabranim ograničenjima, ovisno o potrošenom vremenu ili potrošenoj energiji. Kod funkcije successfulCall može se vidjeti na **Slika 45**.

```
function succesfullCall(response) {
    console.log(response);

    var wkt = response;

    var polygonGeometry = new ol.format.WKT().readGeometry(wkt, {
        dataProjection: 'EPSG:4326',
        featureProjection: 'EPSG:3857'
    });

    console.log(polygonGeometry);

    var polygonFeature = new ol.Feature(polygonGeometry);

    var vectorSource = new ol.source.Vector({
        features: [polygonFeature]
    });

    var colorIndex = lastColorIndex % colorPresets.length;
    lastColorIndex++;

    var color = colorPresets[colorIndex];

    var vectorStyle = new ol.style.Style({
        fill: new ol.style.Fill({
            color: color
        }),
        stroke: new ol.style.Stroke({
            color: color,
            width: 2
        })
    });

    var vectorLayer = new ol.layer.Vector({
        source: vectorSource,
        style: vectorStyle
    });
```

Slika 45. Kod funkcije *succesfullCall*

Također, funkcija *successfulCall* prilikom stvaranja sloja na karti također stvara informacijski prozor na bočnoj traci. Taj prozor omogućava korisniku skrivanje i uklanjanje svih prikazanih slojeva. Također, tipka u prozoru prikazuje vrstu i vrijednost atributa, kao i vrstu grafa koji je prikazan na karti. Kod za stvaranje informacijskog prozora može se vidjeti na **Slika 46**.

```
var pillButton = document.createElement('span');
pillButton.className = 'badge rounded-pill d-flex justify-content-between align-items-center';
pillButton.style.backgroundColor = 'black';
pillButton.style.border = '2px solid ' + color;

var checkmarkIcon = document.createElement('i');
checkmarkIcon.className = 'fas fa-check';
checkmarkIcon.style.color = 'white';
checkmarkIcon.style.marginRight = '5px';

var layerText = document.createElement('span');
if (selectedType == 'time') {
    pillButton.style.width = '140px';
    layerText.innerHTML = 'Time, ' + $('#hidden1').val() + ' min<br>' + KontureType;
    layerText.style.textAlign = 'left';
}
else if (selectedType == 'energy') {
    pillButton.style.width = '155px';
    layerText.innerHTML = 'Energy, ' + $('#hidden2').val() + ' kWh<br>' + KontureType;
    layerText.style.textAlign = 'left';
}
layerText.style.color = 'white';
layerText.style.marginLeft = '0px';

var closeButton = document.createElement('span');
closeButton.textContent = 'x';
closeButton.style.cursor = 'pointer';
closeButton.style.marginLeft = '5px';
closeButton.style.color = 'gray';
```

Slika 46. Kod za stvaranje informacijskog prozora

6.2 Rad web aplikacije

Kako bi korisnik mogao koristiti aplikaciju, potrebno je odabrati željene parametre na bočnoj traci. Izgled bočne trake prikazan je na **Slika 47**.

Slika 47. Izgled bočne trake

Za korištenje aplikacije korisnik prvo treba odabrati vrstu atributa, ovisno o tome želi li vidjeti najkraće rute temeljene na vremenu putovanja ili na energetskoj potrošnji. Nakon odabira vrste atributa, korisnik treba izabrati vrstu grafa koji će prikazati najkraću rutu, s opcijom odabira između konkavnog i konveksnog grafa. Zatim, korisnik treba unijeti vrijednosti za odabrani atribut. Vrijednosti za vrijeme putovanja su u rasponu od 1 do 20 minuta, dok su vrijednosti za energetsku potrošnju u rasponu od 0,1 do 2 kilovat-sata.

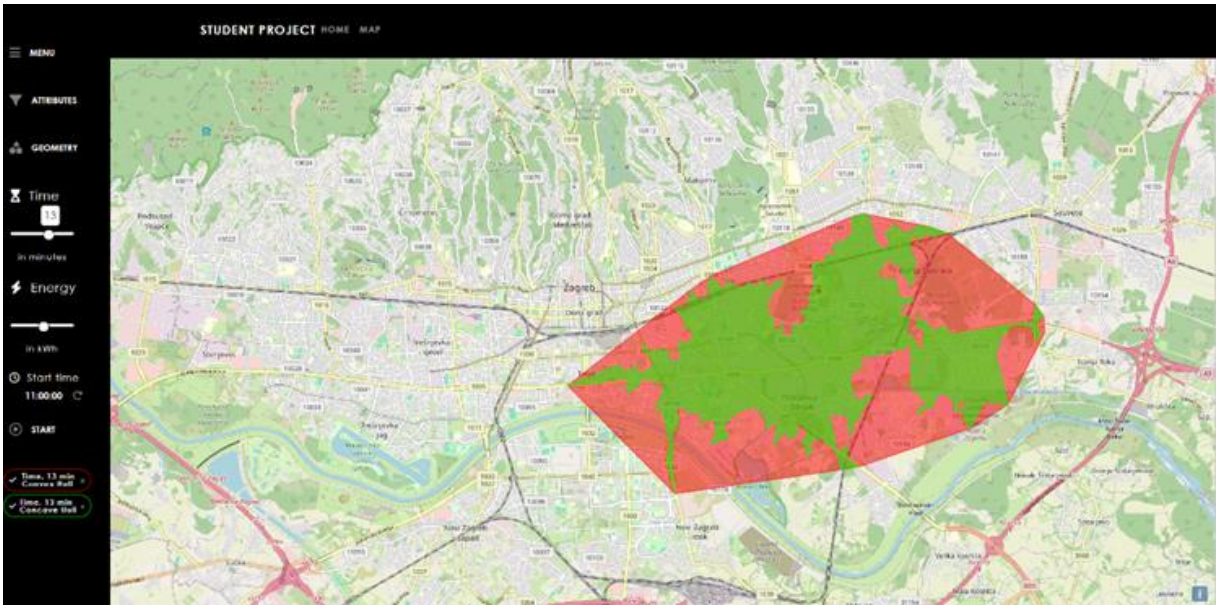
U slučaju da korisnik ne unese vrstu atributa, vrijednosti atributa ili vrstu grafa, prikazuje se prozor upozorenja koji obavještava korisnika da nisu uneseni svi potrebni parametri. Prozor upozorenja prikazan je na **Slika 48**.

Slika 48. Prozor za upozoravanje korisnika

Nakon odabira vrijednosti atributa, korisnik ima mogućnost unosa vremena početka putovanja. Zatim, odabire lokaciju na karti dvostrukim pritiskom na željenu lokaciju na OSM karti. Prozorčić koji prikazuje koordinate odabrane lokacije može se vidjeti na **Slika 49**.

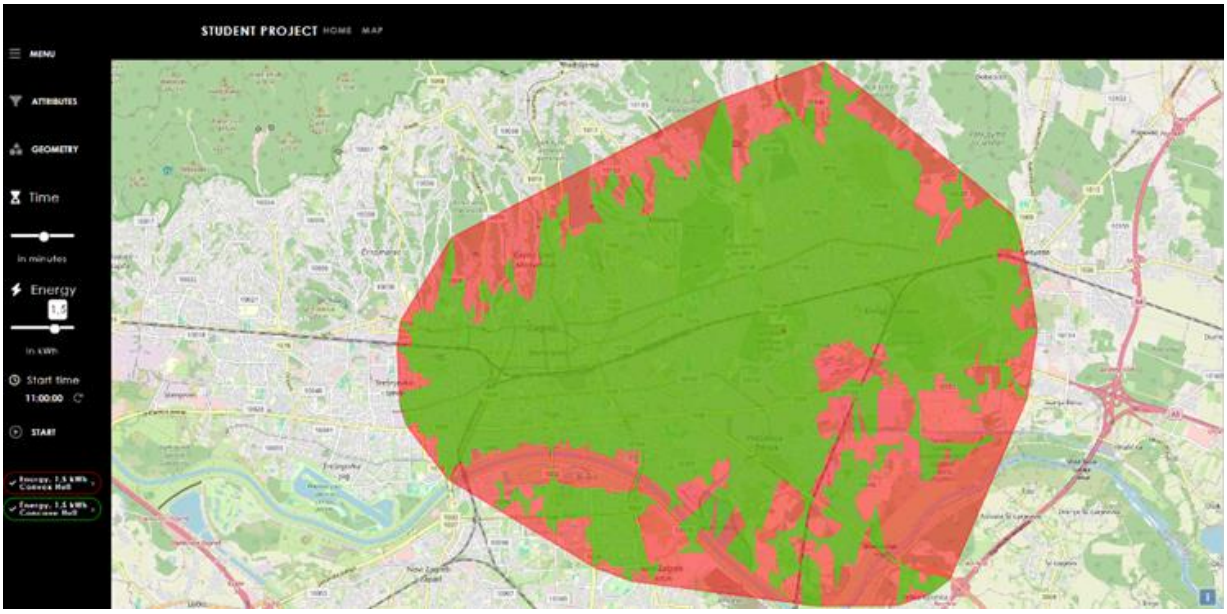
Slika 49. Prozor za ispis koordinata željene lokacije

Nakon što korisnik odabere željenu lokaciju i sve ostale parametre, pritišće tipku *Start*. Tada se prikazuje prostorni doseg prema odabranim parametrima. Za demonstraciju rada web aplikacije korišten je kampus Borongaj. Na **Slika 50** prikazani su konkavni i konveksni grafovi za početnu točku u kampusu Borongaj, s početnim vremenom od 11 sati i vremenom putovanja od 13 minuta. Konveksni graf označen je crvenom bojom, dok je konkavni graf označen zelenom bojom.



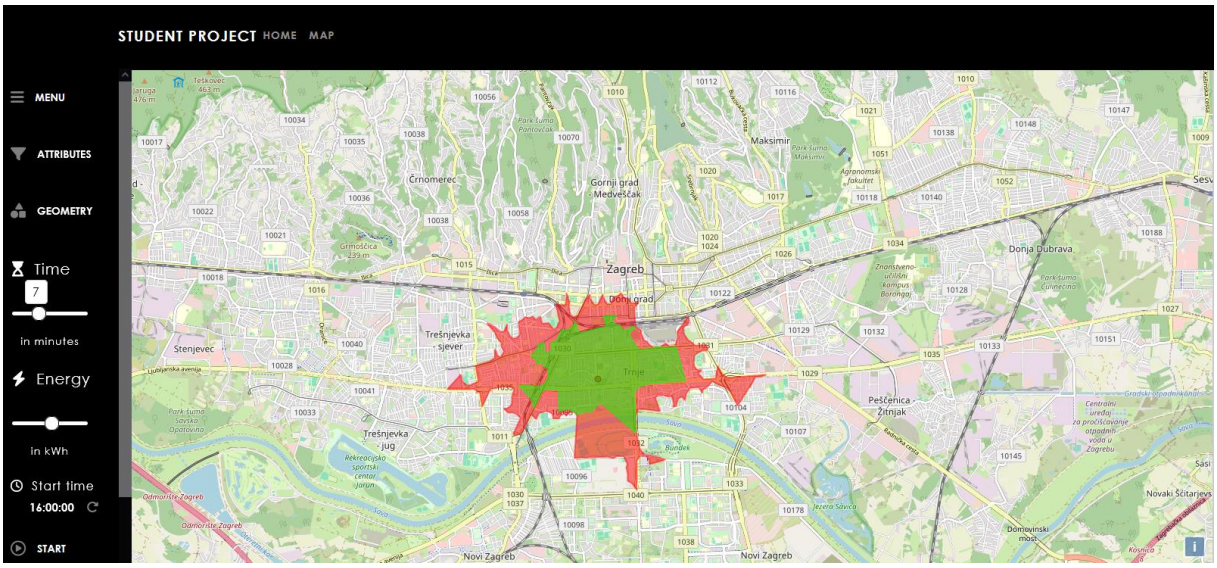
Slika 50. Konkavan i konveksni graf najkraće rute u vremenu putovanja 13 minuta

Na **Slika 51** prikazani su konkavni i konveksni grafovi za scenarij s početnom točkom u kampusu Borongaj, početnim vremenom od 11 sati i potrošenom energijom od 1,5 kilovat-sati. Konveksni graf označen je crvenom bojom, dok je konkavni graf označen zelenom bojom.



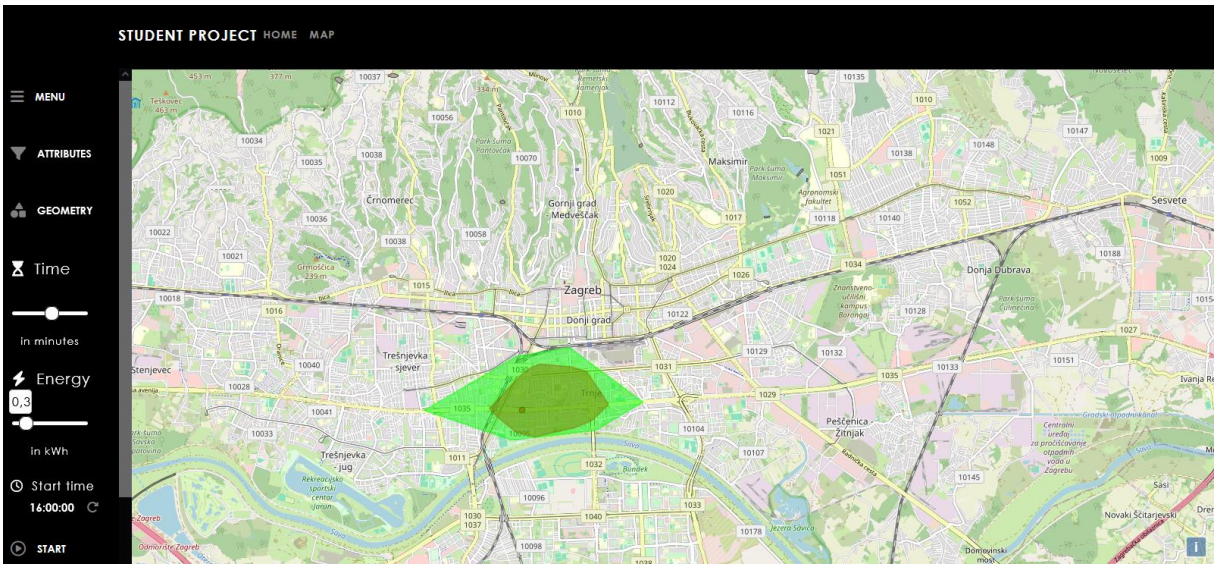
Slika 51. Konkavan i konveksni graf najkraće rute kod potrošnje energije od 1,5 kilovat-sati sati

Grafički prikaz razlike u vremenu putovanja kroz različite periode dana prikazan je na **Slika 52**. U ovoj analizi, za oba scenarija odabran je isti nasumični početak putovanja. Upotrijebljen je konkavni tip grafa, dok je vrijeme dosega postavljeno na 7 minuta. Glavna razlika između dva scenarija nalazi se u vremenu početka putovanja. Prvo putovanje, koje počinje u ponoć (00:00:00), označeno je crvenom konturom, dok drugo putovanje, koje započinje tijekom vršnog sata (16:00:00), označeno je zelenom konturom. Iz prikaza je jasno vidljivo da je područje dosega u zelenoj konturi manje, što potvrđuje informaciju da su brzine vozila tijekom vršnog sata niže, a time su i vremena putovanja duža.



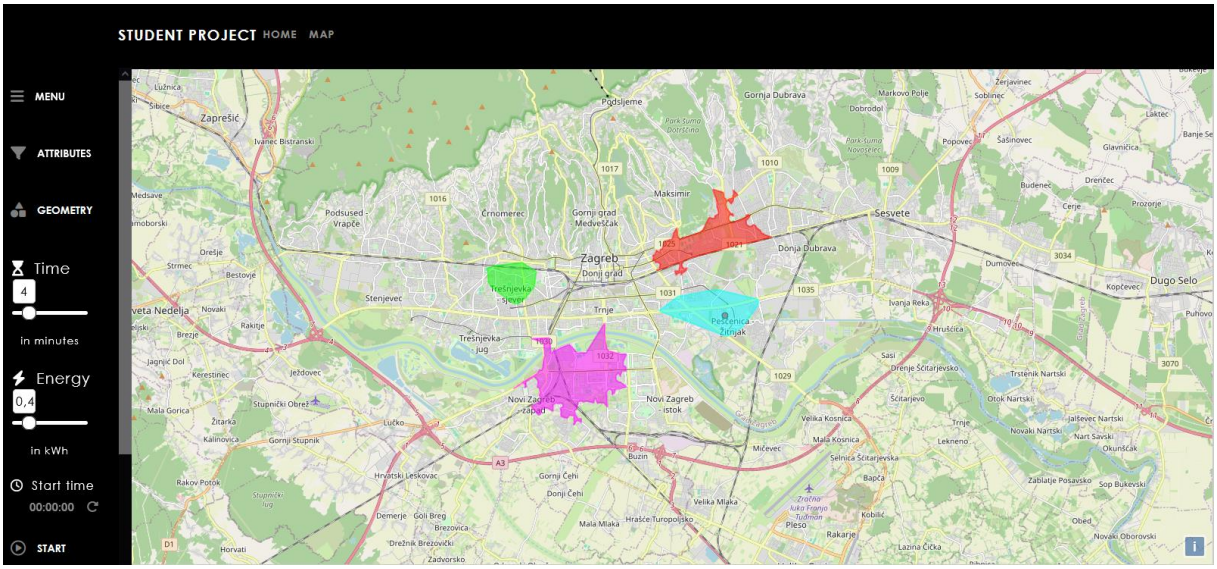
Slika 52. Analiza vremena putovanja ovisno o dobu dana

Sličan primjer prikazuje razliku u potrošnji energije ovisno o dobu dana, kao što je prikazano na **Slika 53**. U ovoj analizi, korišten je konveksni tip grafa, dok je početna lokacija putovanja ista za oba scenarija, ali je odabrana nasumično. Potrošnja energije postavljena je na 0.3 kWh, a vremena početka putovanja su ista kao u prethodnom primjeru - ponoć i vršni sat. Putovanje koje započinje u ponoć označeno je crvenom konturom, dok je putovanje koje započinje tijekom vršnog sata označeno zelenom konturom. U ovom slučaju, može se uočiti da je doseg za vrijeme vršnog sata veći nego u ponoć. To se objašnjava činjenicom da se vozila tijekom vršnog sata kreću sporije i time troše manje energije.



Slika 53. Analiza energetske potrošnje ovisno o dobu dana

Zadnji primjer prikazan na **Slika 54** daje uvid u potpune funkcionalnosti web aplikacije. Aplikacija omogućuje odabir više različitih točaka početka putovanja. Za svako putovanje moguće je izabrati različit tip grafa, različite atribute, njihova ograničenja te različita vremena početka putovanja. Na slici su vidljiva četiri grafa: dva konkavna i dva konveksna. Crveni graf ima doseg od 6 minuta putovanja s vremenom početka putovanja tijekom prvog vršnog sata (07:30:00). Zeleni graf ima energetski ograničen doseg od 0.2 kWh te vrijeme početka putovanja u 19:00:00. Ljubičasti graf također ima energetski ograničen doseg, ali na 0.4 kWh, s vremenom početka putovanja tijekom drugog vršnog sata (16:00:00). Zadnji graf, označen svijetlo plavom bojom, prikazuje vremenski doseg od 4 minute s vremenom početka putovanja u ponoć.



Slika 54. Funkcionalnosti web aplikacije

Izrađena web aplikacija dostupna je za korištenje na službenim stranicama Fakulteta prometnih znanosti (http://fpz.unizg.hr/istenar). Također, snimljen je YouTube video koji demonstrira korištenje i već objašnjene funkcionalnosti cijele web aplikacije (https://www.youtube.com/watch?v=lTmUuGIHs64). Dodatno, cijeli izvorni kod web aplikacije objavljen je na GitHub-u, gdje je dostupan za slobodnu uporabu (https://github.com/terdelic/StudentskiProjektKonture).

7 ZAKLJUČAK

Zaključak ovog istraživačkog rada ističe ključne prednosti primjene inteligentnih transportnih sustava (ITS) u optimizaciji ruta i određivanju prostornog dosega za električna vozila u urbanim područjima, s primjerom grada Zagreba. Korištenjem teorije grafova i sofisticiranih algoritama poput Bellman-Ford i Dijkstrinog algoritma, razvijen je sustav koji omogućuje identifikaciju vremenski najkraćih i energetski optimalnih ruta za električna vozila. Implementacija Fibonaccijeve hrpe ubrzava izvršavanje Dijkstrinog algoritma, dok interaktivno korisničko sučelje, koje uključuje vizualizaciju kontura prostornog dosega, značajno poboljšava praktičnu primjenu ovih algoritama.

Ovaj rad naglašava ključnu ulogu ITS rješenja u unapređenju dosega električnih vozila i planiranju putovanja putem energetski učinkovitijeg usmjeravanja i optimizacije prometa u urbanim područjima, poput Zagreba. Vizualizacija kontura prostornog dosega kroz grafičko sučelje značajno doprinosi boljem razumijevanju i planiranju putovanja. Integracija velikih skupova podataka za realno-vremensko određivanje prostornog dosega značajno pridonosi smanjenju potrošnje energije i povećanju efikasnosti transporta. Za buduća istraživanja od ključne je važnosti kontinuirano usavršavanje sustava, unapređenje tehnika prikupljanja realno-vremenskih podataka te podrška rastućem broju električnih vozila ne samo u Zagrebu, već globalno.

Razvijena web aplikacija u okviru ovog istraživanja može se koristiti putem službenih stranica Fakulteta prometnih znanosti (http://fpz.unizg.hr/istenar). Za detaljniji uvid u funkcionalnosti aplikacije, dostupan je informativni video na YouTube-u (https://www.youtube.com/watch?v=lTmUuGIHs64). Također, cijeli izvorni kod projekta dostupan je na platformi GitHub (https://github.com/terdelic/StudentskiProjektKonture).

Ovaj rad pruža korisne smjernice za razvoj infrastrukture i tehnologije koja podržava održivu mobilnost i smanjenje ekološkog utjecaja prometa. Sustavi poput opisanog imaju značajan potencijal za široku primjenu i daljnji napredak, što će biti ključno za globalnu tranziciju prema održivijim oblicima transporta i urbanog planiranja. U skladu s tim, kontinuirana podrška inovacijama u ITS tehnologijama bit će ključna za postizanje dugoročnih ciljeva održivog razvoja u prometu.

8 ZAHVALE

Zahvaljujemo se našem mentoru, prof. dr. sc. Tončiju Cariću, na strpljenju i vođenju tijekom izrade ovog rada.

Posebno se zahvaljujemo dr. sc. Tomislavu Erdeliću na korisnim savjetima, vrijednom iskustvu i kontinuiranoj podršci. Njegova prisutnost i angažman tijekom cijelog istraživanja bili su ključni za uspješan završetak ovog rada.

Također, zahvaljujemo se Nikoli Mardešiću mag. ing. traff., na korisnoj podršci tijekom razvoja projekta.

Posebno zahvaljujemo kolegici Nikolini Lovrić na pruženoj pomoći i dodatnim doprinosima tekstu rada.

Zahvaljujemo se Zavodu za Inteligentne transportne sustave na njihovoj podršci i pristupu resursima.

LITERATURA

1. Erdelić T. Solving the Electric Vehicle Routing Problem Using a Hybrid Adaptive Large Neighborhood Search Method. [Online].; 2021 [cited 2023 Svibanj. Available from: https://dr.nsk.hr/islandora/object/fpz:2603.

2. Knez D. Rješavanje problema energetski optimalnog puta. [Online].; 2023 [cited 2024. Available from: https://repozitorij.fpz.unizg.hr/islandora/object/fpz%3A2985.

3. Erdelić T, Carić T. A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches. Journal of Advanced Transportation. 2019.

4. Mardešić N. Određivanje energetskog profila prometnice na temelju povijesnih zapisa o kretanju vozila. [Online].; 2019 [cited 2023 Svibanj. Available from: https://dabar.srce.hr/islandora/object/fpz%3A1826.

5. Scaler. Razlika između liste i mape. [Online].; 2022 [cited 2023 Svibanj. Available from: https://www.scaler.com/topics/difference-between-list-and-dictionary-in-python/.

6. Bitnine. Problem sedam mostova Königsberga. [Online].; 2016 [cited 2023 Svibanj. Available from: https://bitnine.net/blog-graph-database/graph-database-and-seven-bridges-of-konigsberg/?ckattempt=2.

7. Gribkovskaia I, Halskau Sr. , Laporte G. The bridges of Königsberg—A historical perspective. Networks. 2007.

8. Wikipedia. Problem sedam mostova Königsberga. [Online].; 2023 [cited 2023 Svibanj. Available from: https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg.

9. Hrvatska enciklopedija. Teorija grafova. [Online].; 2021 [cited 2023 Svibanj. Available from: https://enciklopedija.hr/natuknica.aspx?ID=70127.

10. Carić T. Osnovni pojmovi teorije grafova. [Online].; 2022 [cited 2023 Svibanj. Available from: https://moodle.srce.hr/2022-2023/pluginfile.php/6854518/mod_resource/content/1/01_2012_10_16%20kineski_postar_ispravak_2016.pdf.

11. ResearchGate.. Potpun i nepotpun graf. [Online].; 2019 [cited 2023 Svibanj. Available from: https://www.researchgate.net/figure/A-Incomplete-network-graph-representation-of-observed-breeding-crosses-B-The-complete_fig3_331871062.

12. Saint Louis University. Usmjereni i neusmjereni graf. [Online].; 2020 [cited 2023 Svibanj. Available from: https://cs.slu.edu/~esposito/teaching/1080/webscience/graphs.html.

13. Programiz. Bellman-Ford algoritam. [Online]. [cited 2023 Svibanj. Available from: https://www.programiz.com/dsa/bellman-ford-algorithm.

14. AbuSalim S, Ibrahim R, Saringat MZ, Jamel S, Abdul Wahab J. Comparative Analysis between Dijkstra and. IOP Conference Series: Materials. 2020.

15. Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik. 1959.

16. Wikipedia. Dijkstra algoritam. [Online].; 2023 [cited 2023 Svibanj. Available from: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

17. GeeksForGeeks.. Johnsonova tehnika. [Online].; 2023 [cited 2023 Svibanj. Available from: https://www.geeksforgeeks.org/johnsons-algorithm/.

18. Tarjan RE, Fredman ML. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM. 1987.

19. Programiz. FibonacciHeap. [Online]. [cited 2023 Svibanj. Available from: https://www.programiz.com/dsa/fibonacci-heap.

20. YouTube. FibonacciHeap. [Online].; 2022 [cited 2023 Svibanj. Available from: https://www.youtube.com/watch?v=6JxvKfSV9Ns&list=WL&index=11&.

21. Programiz. FibonacciHeap - Decrease Key. [Online]. [cited 2023 Svibanj. Available from: https://www.programiz.com/dsa/decrease-key-and-delete-node-from-a-fibonacci-heap.

22. GeeksForGeeks. MVC Framework Introduction. [Online].; 2023 [cited 2023 Srpanj. Available from: https://www.geeksforgeeks.org/mvc-framework-introduction/.

23. GeeksForGeeks. What is NuGet. [Online].; 2020 [cited 2023 Srpanj. Available from: https://www.geeksforgeeks.org/what-is-nuget/.

SAŽETAK

Dominik Knez, Marko Kukor, Ante Dumančić

Vizualizacija najkraćih ruta vremena putovanja i energetske potrošnje

U ovom radu razvijen je inovativni sustav za analizu prometnih tokova s naglaskom na određivanje prostornog dometa vozila prema vremenskim i energetskim ograničenjima. Koristeći podatke prikupljene iz projekta SORDITO, modelirana je stvarna cestovna mreža kao graf, omogućavajući primjenu modificiranih Dijkstrinih i Bellman-Fordovih algoritama. Implementacija Fibonacci heap strukture značajno je poboljšala performanse algoritama, što je ključno za brzu analizu velikih skupova podataka.

Kao rezultat istraživanja, razvijeno je interaktivno web grafičko sučelje koje omogućuje prikaz kontura prostornog dosega vozila u odnosu na vremenska ili energetska ograničenja. Korisnici mogu vidjeti, primjerice, do kojih odredišta mogu stići unutar 10 minuta ili s potrošnjom od 1 kWh počevši od određenog početnog mjesta i vremena. Vizualizacija se temelji na poligonima koji jasno prikazuju doseg na karti, što olakšava donošenje informiranih odluka o putovanjima. Razvijena aplikacija dostupna je za korištenje putem službenih stranica Fakulteta prometnih znanosti (http://fpz.unizg.hr/istenar).

Ovaj rad ističe važnost integracije različitih izvora podataka i primjenu naprednih algoritama za efikasno rješavanje problema optimizacije u stvarnom vremenu, uz posebno naglašavanje uloge vizualizacije rezultata. Kombinacija teorijskog pristupa i praktične implementacije kroz interaktivno sučelje pruža korisnicima intuitivan alat za planiranje ruta, čime se doprinosi unapređenju mobilnosti i efikasnosti u urbanim područjima.

Ključne riječi: Vremenski najkraći put; Energetski optimalni put; Dijkstra algoritam; Fibonacci heap struktura podataka; Konture prostornog dosega

SUMMARY

Dominik Knez, Marko Kukor, Ante Dumančić

Visualization of shortest travel time and energy consumption routes

This research introduces an innovative system for traffic flow analysis, focusing on determining the spatial range of vehicles based on time and energy constraints. Utilizing data collected from the SORDITO project, the real road network has been modeled as a graph, enabling the application of modified Dijkstra's and Bellman-Ford algorithms. The implementation of the Fibonacci heap structure has significantly enhanced algorithm performance, which is crucial for rapid analysis of large datasets.

As a result of the research, an interactive web graphical interface has been developed, allowing the display of isochrones (contours of vehicle spatial reach) in relation to time or energy constraints. Users can, for example, see which destinations they can reach within 10 minutes or with a consumption of 1 kWh starting from a given initial location and time. The visualization is based on polygons that clearly depict the isochrones on the map, facilitating informed travel decisions. The developed application is available for use through the official website of the Faculty of Transport Sciences (http://fpz.unizg.hr/istenar).

This paper highlights the importance of integrating various data sources and applying advanced algorithms for efficient real-time optimization, with a particular emphasis on the role of result visualization. The combination of theoretical approach and practical implementation through an interactive interface provides users with an intuitive tool for route planning, contributing to the enhancement of mobility and efficiency in urban areas.

Keywords: Shortest time path; Energy optimal path; Dijkstra algorithm; Fibonacci heap data structure; Isochrones

ŽIVOTOPIS AUTORA

Dominik Knez rođen je 29. travnja 2000. godine u Zagrebu, Hrvatska. Godine 2019. završio je svoje srednjoškolsko obrazovanje u Gornjogradskoj gimnaziji u Zagrebu, opći smjer. Godine 2023. završio je preddiplomski studij Inteligentni transportni sustavi i logistika na Fakultetu prometnih znanosti Sveučilišta u Zagrebu, smjer Inteligentni transportni sustavi. Trenutno je na diplomskom studiju na istom smjeru na istom fakultetu.

Marko Kukor rođen je 27. rujna 2001. godine u Zagrebu, Hrvatska. Godine 2020. završio je svoje srednjoškolsko obrazovanje u Gimnaziji Velika Gorica u Velikoj Gorici, opći smjer. Godine 2023. završio je preddiplomski studij Inteligentni transportni sustavi i logistika na Fakultetu prometnih znanosti Sveučilišta u Zagrebu, smjer Inteligentni transportni sustavi. Trenutno je na diplomskom studiju na istom smjeru na istom fakultetu.

Ante Dumančić rođen je 25. travnja 2001. godine u Kninu, Hrvatska. Godine 2020. završio je svoje srednjoškolsko obrazovanje u Srednjoj školi Kupres u Kupresu, opći smjer. Godine 2023. završio je preddiplomski studij Inteligentni transportni sustavi i logistika na Fakultetu prometnih znanosti Sveučilišta u Zagrebu, smjer Inteligentni transportni sustavi. Trenutno je na diplomskom studiju na istom smjeru na istom fakultetu.

POPIS SLIKA

POPIS TABLICA